NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

RECEIVER OPERATING CHARACTERISTIC CURVES FOR LINEAR ARRAYS OF VECTOR SENSORS USING NONLINEAR CARDYNULL PROCESSING

by

David Tassia

March 2011

Thesis Co-Advisors: Kevin B. Smith
Lewis Meier

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2011	Master's Thesis

4. TITLE AND SUBTITLE
Receiver Operating Characteristic Curves for Linear Arrays of Vector Sensors Using Nonlinear Cardynull Processing

5. FUNDING NUMBERS

6. AUTHOR(S) David Tassia

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	A

13. ABSTRACT (maximum 200 words)

During the past decade researchers have been considering vector sensors for use in linear towed arrays for passive target detection. Linear processing is often used due to its simplicity and significant directivity improvements. Nonlinear processing holds the potential for further directivity improvements; however, it also presents the risk of amplifying uncorrelated noise.

This thesis simulated a correlated signal in uncorrelated noise to investigate the potential of a nonlinear (but non-adaptive) processing technique. It demonstrates that the increased directivity and substantially diminished response from the ambiguous direction is quite beneficial when the signal is located within certain quadrants. It also demonstrates that linear processing is more effective than this nonlinear processor near endfire. In all cases, the signal to noise ratio was high enough to be detectable by basic array gain from multiple sensors.

Monte Carlo simulations were completed to generate detection statistics and ROC curves were created to illustrate the relative effectiveness of: pressure-only sensor arrays, linearly processed vector sensor arrays, and nonlinearly processed vector sensor arrays. For a broadside signal in uncorrelated noise, simulations indicate an array with eleven vector sensors can achieve a 3 dB improvement if the nonlinear processing defined in this thesis is utilized instead of linear processing.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Nonlinear, Non-Adaptive Processing, Receiver Operating Characteristic (ROC) Curves, Vector Sensors, Towed Array, Passive Detection	115
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

RECEIVER OPERATING CHARACTERISTIC CURVES FOR LINEAR ARRAYS OF VECTOR SENSORS USING NONLINEAR CARDYNULL PROCESSING

David Tassia
Naval Undersea Warfare Center Division Newport
B.S., University of Massachusetts–Lowell, 1984
M.S., Mechanical Engineering, Catholic University of America, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING ACOUSTICS

from the

NAVAL POSTGRADUATE SCHOOL
March 2011

Author: David Tassia

Approved by: Kevin B. Smith
Thesis Co-Advisor

Lewis Meier
Thesis Co-Advisor

Daphne Kapolka
Chair, Engineering Acoustics Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

During the past decade researchers have been considering vector sensors for use in linear towed arrays for passive target detection. Linear processing is often used due to its simplicity and significant directivity improvements. Nonlinear processing holds the potential for further directivity improvements; however, it also presents the risk of amplifying uncorrelated noise.

This thesis simulated a correlated signal in uncorrelated noise to investigate the potential of a nonlinear (but non-adaptive) processing technique. It demonstrates that the increased directivity and substantially diminished response from the ambiguous direction is quite beneficial when the signal is located within certain quadrants. It also demonstrates that linear processing is more effective than this nonlinear processor near endfire. In all cases, the signal to noise ratio was high enough to be detectable by basic array gain from multiple sensors.

Monte Carlo simulations were completed to generate detection statistics and ROC curves were created to illustrate the relative effectiveness of pressure-only sensor arrays, linearly processed vector sensor arrays, and nonlinearly processed vector sensor arrays. For a broadside signal in uncorrelated noise, simulations indicate an array with eleven vector sensors can achieve a 3 dB improvement if the nonlinear processing defined in this thesis is utilized instead of linear processing.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

dB	Decibels
CW	Continuous Wave
NPS	Naval Postgraduate School
NUWC	Naval Undersea Warfare Center
Own Ship	The surface ship or submarine that deploys the towed array.
ROC	Receiver Operating Characteristic
S&N	Signal and Noise
SNR	Signal to Noise Ratio
U-boat	German Submarine
VTC	Video Teleconferencing Center

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to acknowledge the assistance of my advisors, Professor Kevin Smith, PhD (Naval Postgraduate School) and Lewis Meier, PhD (Naval Undersea Warfare Center, Newport Division), in the preparation of this thesis. Notably, I credit Dr. Smith with the concept and naming of cardynull processing described in detail in this thesis. I also greatly appreciate the expertise and advice of Dr. Meier regarding the statistical detection aspects of this thesis including the preparation of ROC curves. Without periodic consultation with both of my advisors, this thesis might never have been completed.

I also appreciate everyone involved in making Engineering Acoustics Distance Learning program possible. While I have never been in the same room with many of my professors, they have helped me greatly through their prepared lectures and assignments. I appreciate the diligent efforts of Video Teleconferencing Center (VTC) personnel at both the Naval Undersea Warfare Center (NUWC) and the Naval Postgraduate School (NPS) that consistently provided the critical link that allowed us to view and participate in NPS lectures over a three-year period.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Research and development of sensor arrays towed behind ships for passive submarine detection began as early as World War I (Urick, 1983) as a means to overcome the U-boat threat. After the war, the development of the "eel," an array of 12 pressure sensors, was documented (Hayes, 1920). This system contains many of the characteristics of a modern "towed array" system. In the current decade, researchers have considered the use of vector sensors[1] to measure both pressure and particle velocity at the array. These systems provide improved directivity (Cray & Nuttall, 2001) resulting in diminished target ambiguity.

B. RESEARCH MOTIVATION

Recent efforts have suggested the use of unconventional nonlinear processing techniques (Cox & Zeskind, 1992; Smith & Leijen, 2007) to foster further improvements in array directivity. The benefits of increased directivity are clear; however, it was not clear if the proposed nonlinear processing would lead to amplification of uncorrelated noise. The effort described in this thesis addresses this uncertainty by simulating both the signal and noise while providing graphics to show how the processing techniques compare.

C. RESEARCH SUMMARY

A correlated signal in uncorrelated noise is simulated and Receiver Operating Characteristic (ROC) Curves are plotted to demonstrate the classification effectiveness of various processing techniques. These techniques include conventional linear processing of pressure-only arrays, linear cardioid processing of vector sensor arrays, and a new nonlinear (but non-adaptive) technique for processing of vector sensor arrays. The results show the new

[1] Vector sensors include a pressure element and three orthogonal acceleration elements.

nonlinear processing technique provides improved detection performance over the conventional linear processing techniques provided the signal is located within the broadside quadrant. It should be noted, however, that this thesis only considered signals that had large enough signal-to-noise to be detectable by basic array gain due to multiple sensors. The issue of detection of much lower signals is not addressed.

II. THEORY

A. ACOUSTICS AND SIGNAL PROCESSING

1. Coordinate System

This thesis uses the rectangular coordinate system shown in Figure 1. Linear arrays discussed here will be oriented along the x-axis as shown. The orientation of incoming planar waves to this array will be defined using the angles θ and ϕ. Steering angles will be defined using similar angles θ_s and ϕ_s. As shown in the figure, angles θ and θ_s represent roll angles and the angles ϕ and ϕ_s rotate around an axis perpendicular to the roll direction. For simplicity, this thesis considers roll angles θ and θ_s equal to zero. Consequently, the angles ϕ and ϕ_s are positive rotations around the z-axis. The angles ϕ and ϕ_S may also be referred to as the bearing angle and the steering angle, respectively.

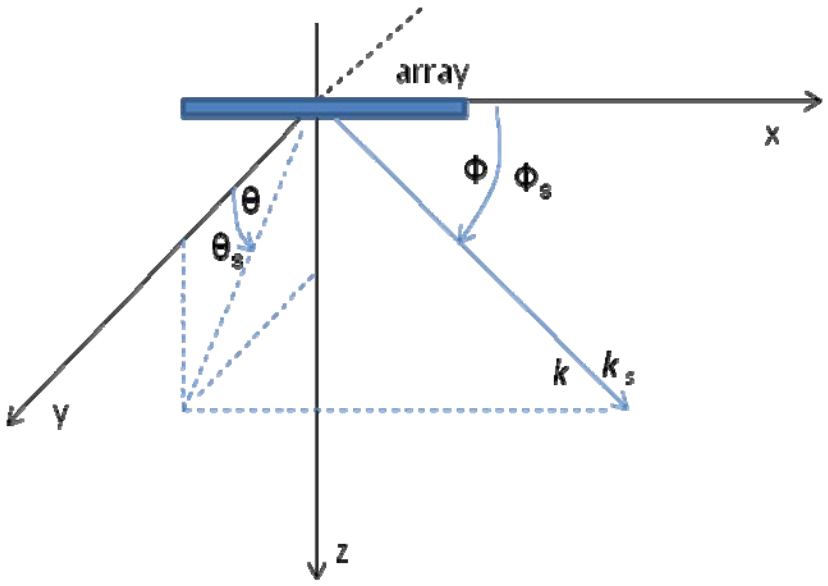

Figure 1 Coordinate System for This Thesis

2. Sound Transmission in Water

Sound travels through water in longitudinal waves. These waves include local compressions and rarefactions of the water that can travel for long distances. Once these waves contact a receiver, such as a hydrophone, the pressure fluctuations will cause the receiver to vibrate resulting in measured sound signal at the receiver.

Plane wave propagation in homogeneous isotropic fluids is given by (Kinsler et al., 1976)

$$p(x,y,z,t) = Ae^{-i(\omega t - \vec{k}\cdot\vec{r})}, \tag{1}$$

where

$p(x,y,z,t)$ = acoustic pressure,

$A = amplitude$,

$\omega =$ angular frequency $= 2\pi f$,

$\vec{k} =$ propagation wavenumber vector $= k_x\hat{i} + k_y\hat{j} + k_z\hat{k}$ (with magnitude ω / c),

$\vec{r} =$ position vector $= x\hat{i} + y\hat{j} + z\hat{k}$ (from the origin of the coordinate system), and

$\hat{i}, \hat{j}, \hat{k}$ = unit vectors in the x,y,z directions, respectively.

This thesis will focus on the spatial processing provided by linear arrays and not temporal processing techniques. Consequently, this equation is simplified to

$$p(x,y,z) = Ae^{i(\vec{k}\cdot\vec{r})}. \tag{2}$$

3. Particle Velocity

For a plane wave, particle velocity is related to pressure by

$$v = \frac{p}{Z} \tag{3}$$

where

$$Z = \pm \rho c$$

and

$\rho =$ density of the undersea medium,
$c =$ speed of sound in the undersea medium,
$Z =$ characteristic impedance of the undersea medium, and
the sign $(\pm)$ is determined by the direction of propagation.

Consequently, the pressure is directly proportional to the velocity scaled by an amount equal to the characteristic impedance.

Particle velocity is typically obtained indirectly using accelerometers. Considering a broadside (y-direction) planar wave, particle velocity and the acceleration measured are related by

$$a_y = \frac{\partial v_y}{\partial t} \tag{4}$$

where

$a_y =$ acceleration measured in the y-direction.

For a wave with a single frequency, ω, this reduces to a phase shift, since

$$a_y = \frac{\partial v_y}{\partial t} = -i\omega v_y \Rightarrow v_y = i\frac{a_y}{\omega} = \frac{a_y}{\omega} e^{i\pi/2}. \tag{5}$$

4. Vector Sensor

A vector sensor is a hydrophone that includes a pressure element and a tri-axial accelerometer collocated at a single point (Cray, 2002; Cray & Evora, 2004). The pressure element will produce an omni-directional response as shown in Figure 2. An accelerometer element facing toward 90° will produce a response as shown in Figure 3. Integration of the measured response as described by the previous equation will also result in the same pattern.

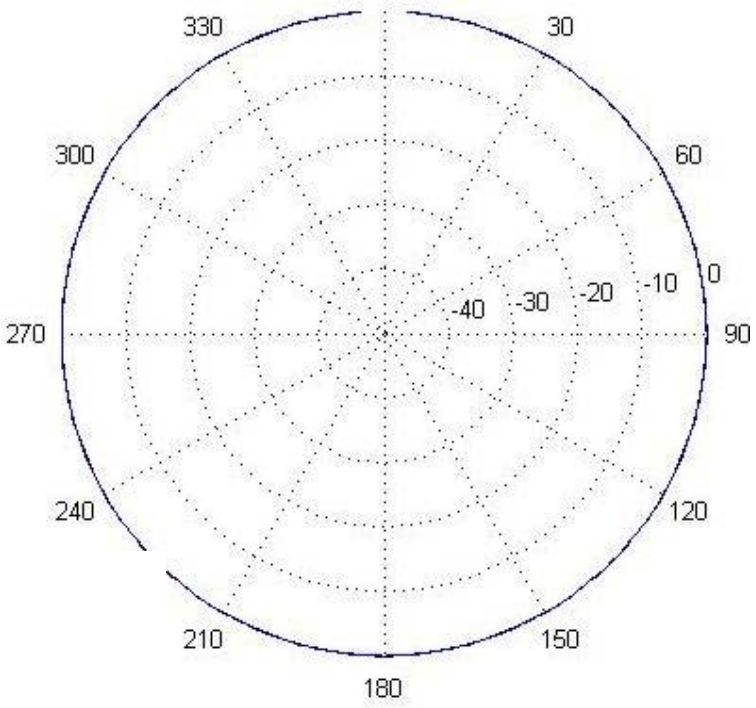

Figure 2 Polar Plot of Omni-directional Hydrophone

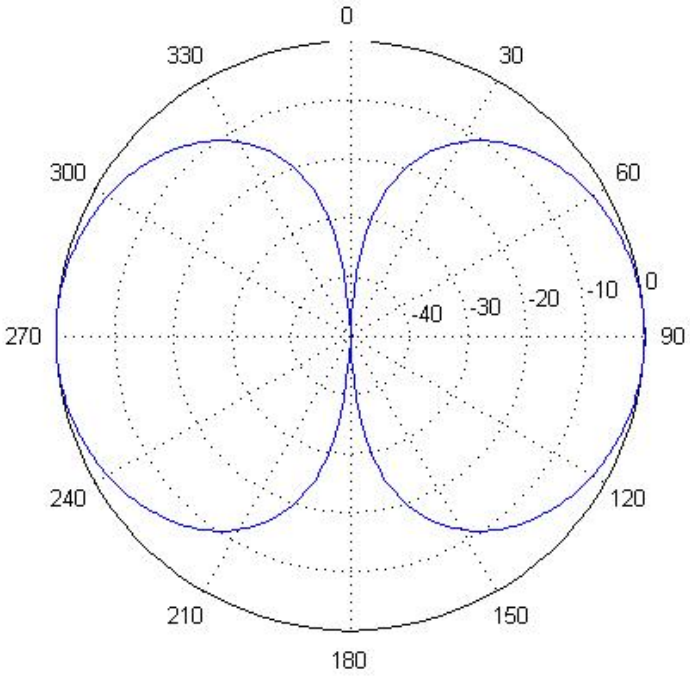

Figure 3 Dipole Pattern from Accelerometer Oriented toward 90°

It is possible to combine the responses of the previous two figures to obtain the pattern shown in Figure 4. Traditionally this pattern has been called the “cardioid” pattern since it is similar to a heart shape. As shown, it preferentially detects sound from 90° while rejecting sound from 270°. This is advantageous for determining the location of a signal.

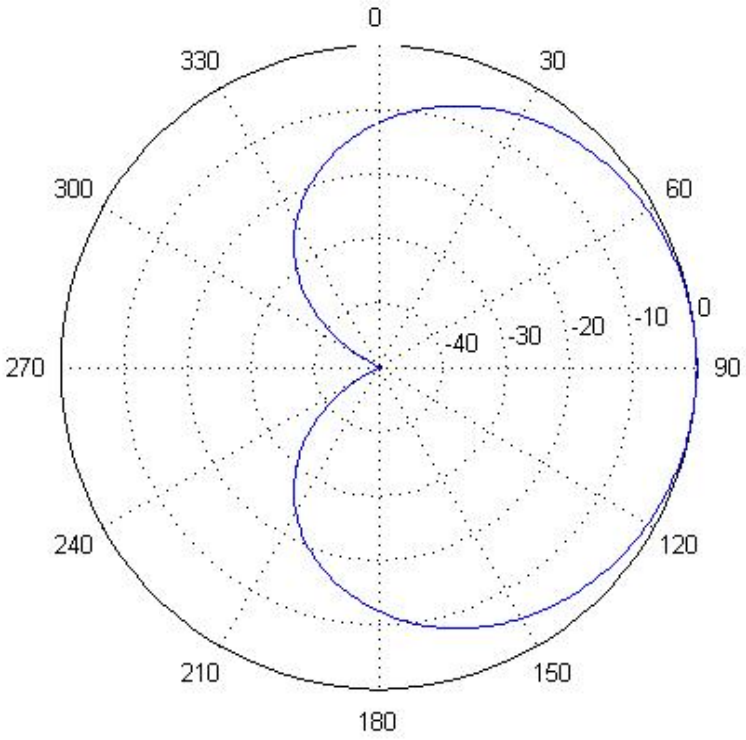

Figure 4 Cardioid Beampattern from Omni-directional and Dipole

The pattern shown in the previous figure can be obtained by the simple sum of a single pressure element and the response of a single acceleration element (converted to velocity) oriented toward the 90° direction. This configuration, however, limits the orientation of the cardioid to the direction shown. This is why vector sensors typically incorporate three perpendicular axial accelerometers in addition to the pressure element. The linear summation of all of these elements can create a three-dimensional cardioid response that can be steered to obtain the maximum response in any direction.

5. Linear Arrays

Sensors are often arranged in arrays to improve signal detection. This thesis will consider linear arrays. A typical arrangement involving pressure sensors is shown in Figure 5. A similar arrangement can also be created using vector sensors.

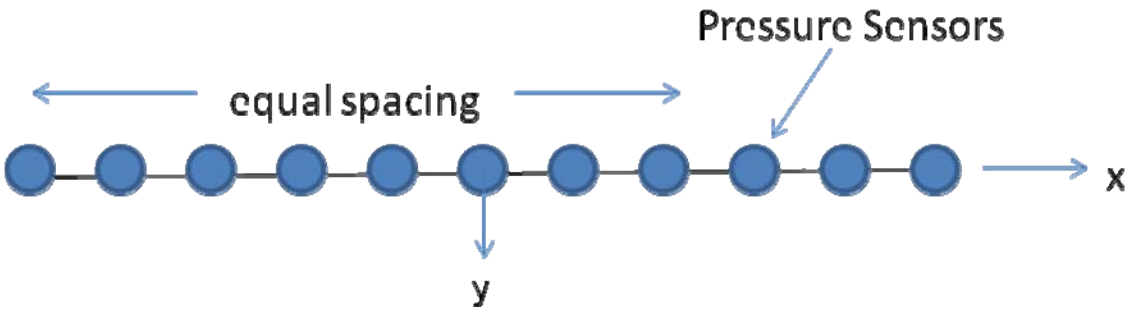

Figure 5 Typical Linear Array Arrangement

A linear array will amplify responses from a particular direction thereby favoring a signal in that direction over noise that comes from all directions. This preferential treatment of the correlated signal over uncorrelated noise is the basis for effective array processing. This paper will concentrate on linear arrays that are commonly towed behind ships for passive detection of submarines. The focus will be on the spatial processing provided by the arrays and not any temporal processing techniques.

Array gain (AG) is defined by (Urick, 1983)

$$AG = 10\log\frac{(S/N)_{array}}{(S/N)_{element}} \tag{6}$$

where

$(S/N)_{array}$ = Signal Power to Noise Power Ratio for the Array, and

$(S/N)_{element}$ = Signal Power to Noise Power Ratio for One Sensor.

The numerator typically will be larger than the denominator since the array will amplify the response in a given direction while diminishing responses from other directions. This arrangement favors the signal since it has a single location while noise originates from all directions.

6. Beamforming

Beamforming is commonly used with sensor arrays to amplify sound from a particular direction, while diminishing sound from other directions. The technique is achieved by summing the response from each sensor. The listening direction can be chosen by adjusting the timing delay or phase shift between the individual sensors.

a. Pressure-Only Sensor Array

Figure 5 illustrates the basic pressure array arrangement. Each pressure sensor has an omni-directional response as shown in Figure 2; however, if an array of eleven pressure sensors are oriented along the 0–180° and summed the response to a broadside signal is shown in Figure 6.

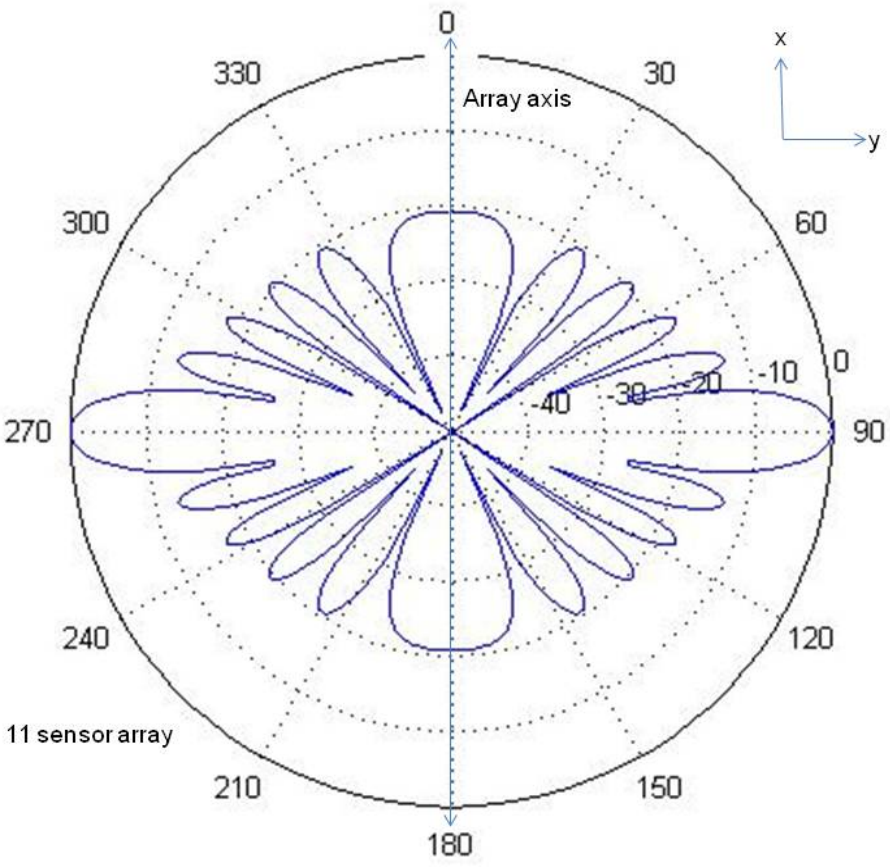

Figure 6 Two-Dimensional[2] Linear Beampattern for a Pressure Sensor Array Due to Incoming Signal at Broadside

Comparison of Figure 2 and Figure 6 demonstrates the value of arrays: the array (Figure 6) sound from the broadside direction (90° and 270°) is louder than other directions, while the single pressure sensor (Figure 2) "hears" sound equally from all directions. Since arrays can selectively listen, the direction to a signal can be established.

[2] Throughout this thesis, all two-dimensional polar plots have angles in degrees measured clockwise around the center of the array with zero representing forward endfire. Each circle within the polar plot is labeled in decibels relative to the maximum response (0 dB).

The equation for beamforming an array of pressure-only sensors is

$$B_{linear}^{(p)}(\theta_s,\phi_s) = \left| \sum_n (w_{pn} P_n) e^{-ikx_n \cos\phi_s} \right|^2 \quad (7)$$

where

$B_{linear}^{(p)}(\theta_s,\phi_s)$ = linear beampattern for pressure sensor array,

w_{pn} = weighting factors,

P_n = the pressure wave at each sensor, and

ϕ_s = the steering direction.

The beampattern shown in Figure 6 is illustrated in the three-dimensional plot shown in Figure 7. The three-dimensional plot shows that the lobes are symmetric around the array axis.

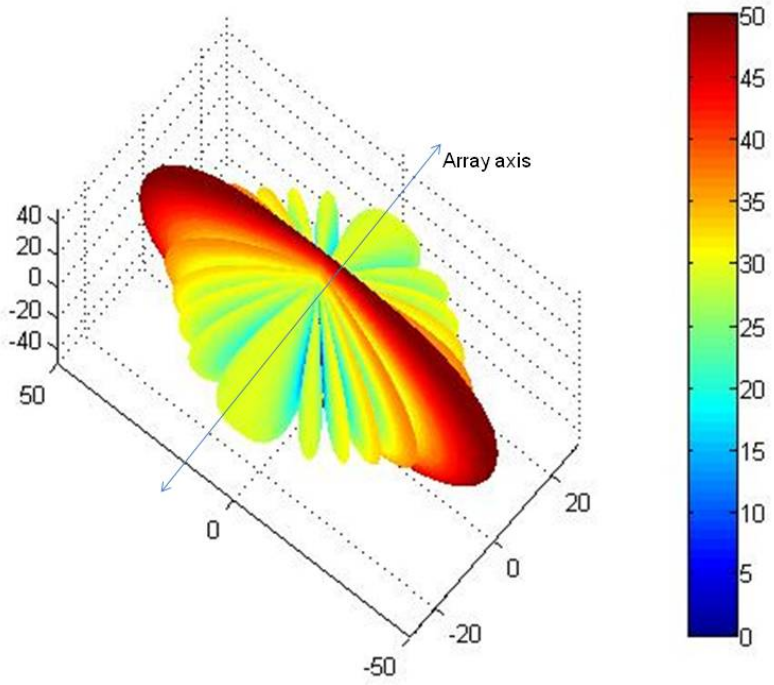

Figure 7 Three-Dimensional Linear Beampattern for a Pressure Sensor Array Due to Incoming Signal at Broadside

This symmetric behavior is common for pressure arrays regardless of the location of the signal. For example, the beampattern for the same array with the signal located 45° aft of broadside is shown in Figure 8 and Figure 9.

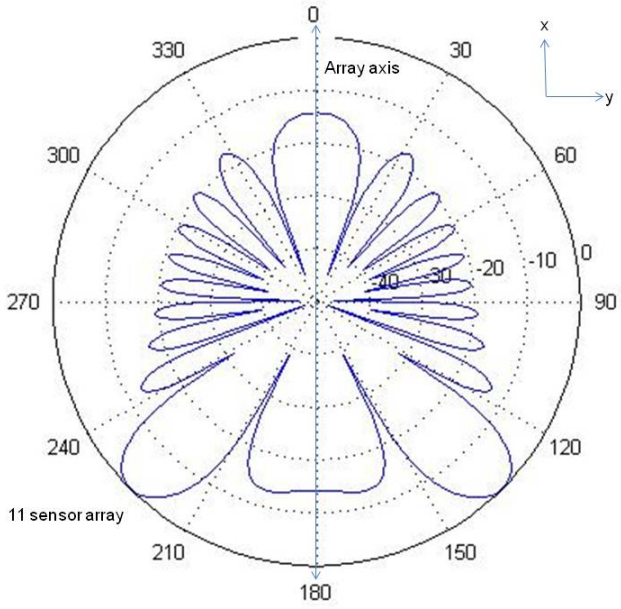

Figure 8 Two-Dimensional Linear Beampattern for a Pressure Sensor Array Due to Incoming Signal at 135°

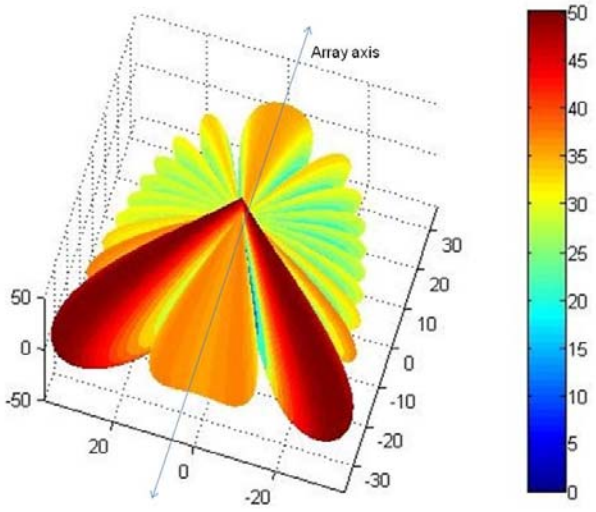

Figure 9 Three-Dimensional Linear Beampattern for a Pressure Sensor Array Due to Incoming Signal as 135°

The symmetric beampatterns of Figure 7 and Figure 9 illustrate the primary limitation of pressure sensor arrays. By steering the array, it is possible to increase the received signal (from a target of interest) relative to ambient noise. When the signal to noise is maximized, we can speculate that the direction to the target is likely located in one of the largest lobes (see Figure 7 and Figure 9); however, without additional information, we do not know in which of the ambiguous directions the target is located. Location ambiguity is common in passive sonar and Figure 10 illustrates what has been called the cone of ambiguity. For passive detection of a signal using only a pressure sensor array, we would anticipate only determining that the signal is somewhere on the cone of ambiguity. Additional information is not available without maneuvering the array or adding additional sensors.

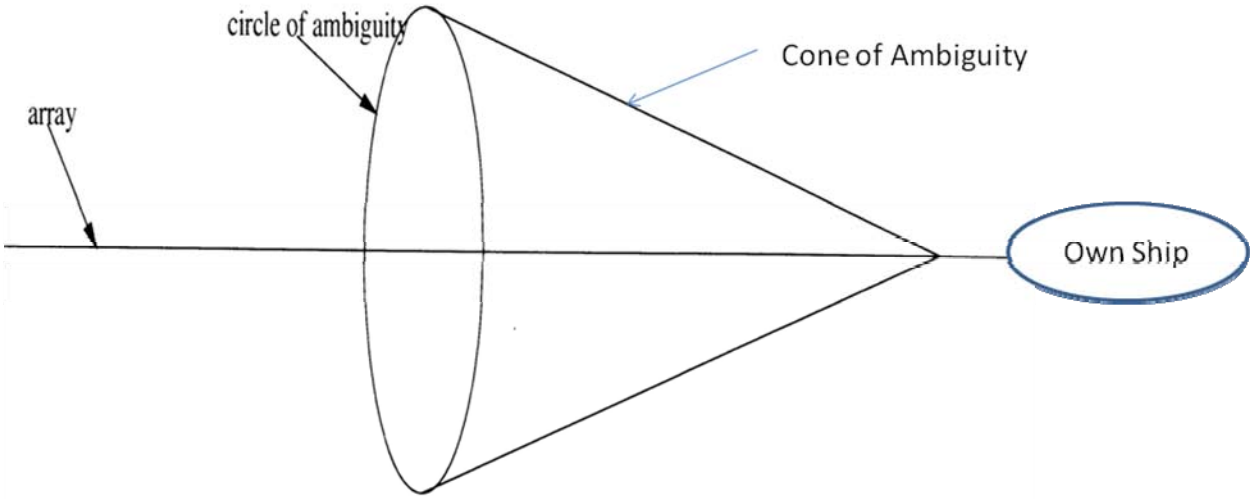

Figure 10 Cone of Ambiguity for a Typical Towed Array

Sometimes the ambiguity described above is considered in a two-dimensional sense and may, therefore, be referred to as left/right ambiguity.

b. Vector Sensor Array—Linear Processing

Beamforming can be accomplished for an array of vector sensors by using a process similar to the one described in the previous section. The linear processor is typically defined by the following equations (Smith & Leijen, 2007).

$$B_{linear}^{(pv)}(\theta_s,\phi_s) = \left| \sum_n b_n^{(pv)} e^{-ikx_n \cos\phi_s} \right|^2 \tag{8}$$

where

$$B_{linear}^{(pv)}(\theta_s,\phi_s) = \text{linear beampattern for vector sensor array,}$$
$$\sum_n b_n^{(pv)}(\theta_s,\phi_s) = (w_{xn}v_{xn} + w_{yn}v_{yn} + w_{zn}v_{zn} + w_{pn}v_{pn}),$$
$$w_{xn}, w_{yn}, w_{zn}, w_{pn} \text{ are weighting factors,}$$
$$v_{xn} = V_{xn}e^{i\vec{k}\cdot\vec{r}_n}, v_{yn} = V_{yn}e^{i\vec{k}\cdot\vec{r}_n}, v_{zn} = V_{zn}e^{i\vec{k}\cdot\vec{r}_n}, v_{pn} = V_{pn}e^{i\vec{k}\cdot\vec{r}_n},$$
$$\vec{k} = \text{the wave vector for the propagating wave,}$$
$$\vec{r_n} = \text{sensor position index, and}$$
$$V_{xn} = V_n\cos\phi, V_{yn} = V_n\cos\theta\sin\phi, V_{zn} = V_n\sin\theta\sin\phi, V_{pn} = \frac{P_n}{\rho c}.$$

Typical weighting factors are:

$$w_{xn} = w_n\cos\phi_s,\ w_{yn} = w_n\cos\theta_s\sin\phi_s,\ w_{zn} = w_n\sin\theta_s\sin\phi_s,\ w_{pn} = w_n.$$

where

$$\theta_s, \phi_s = \text{the roll and bearing steering angles}.$$

Note that the variable V_{pn} contains weighting to compensate for the characteristic impedance, ρc, of the undersea medium.

This linear processor will be referred to as cardioid processing here since each sensor (consisting of a pressure element and three axial accelerometers) will produce a cardioid pattern (see Figure 4) when the weights of each component are equal. When the cardioid from each sensor is summed with responses from other sensors in the array, the beampattern shown in Figure 11 is obtained for a broadside signal.

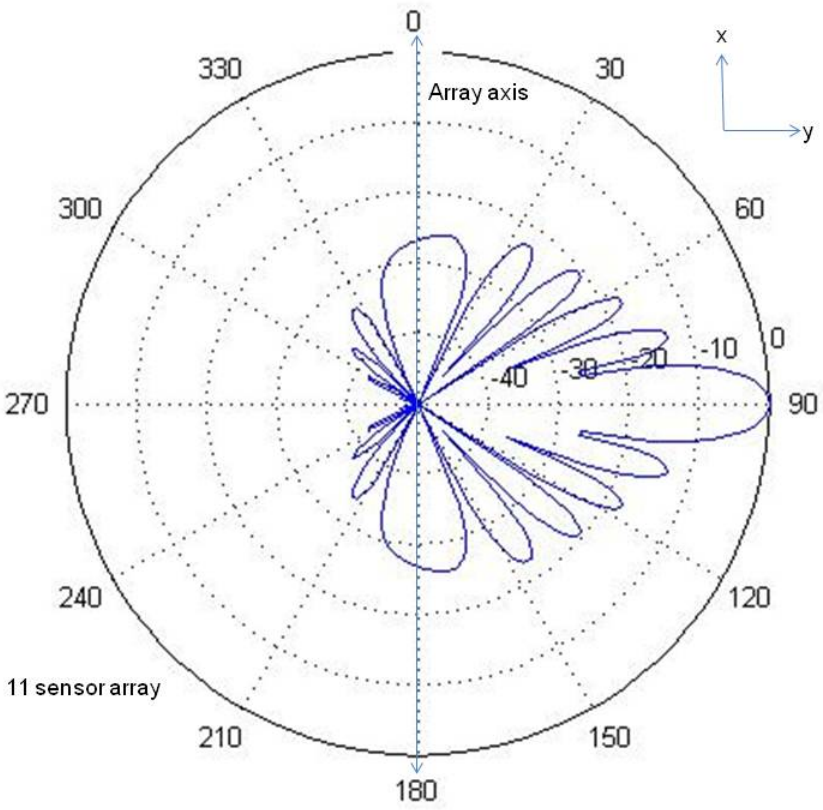

Figure 11 Cardioid Beampattern for a Vector Sensor Array Due to Incoming Signal at Broadside

Comparison of Figures Figure 6 and Figure 11 show the significant advantage of vector sensor arrays as compared to pressure sensor arrays. Figure 6 shows that the pressure array beampattern has a large lobe in the ambiguous direction (i.e., 270°). Figure 11 shows that the vector sensor array does not have a lobe in the ambiguous direction (i.e., 270°) for the same case. Consequently, we would anticipate the ship operating a vector sensor array would require less maneuvering than a similar vessel operating a pressure sensor array.

Although Figure 11 showed perfect rejection of sound from the ambiguous direction, the performance is not as good when the array is steered to a direction other than broadside. For example, for an incoming signal at 135°, the beampattern shown in Figure 12 is obtained. Although this pattern does not

provide perfect rejection of sound in the ambiguous direction, it provides a clear improvement (around 6 dB) over a similar case for a pressure sensor array (see Figure 8).

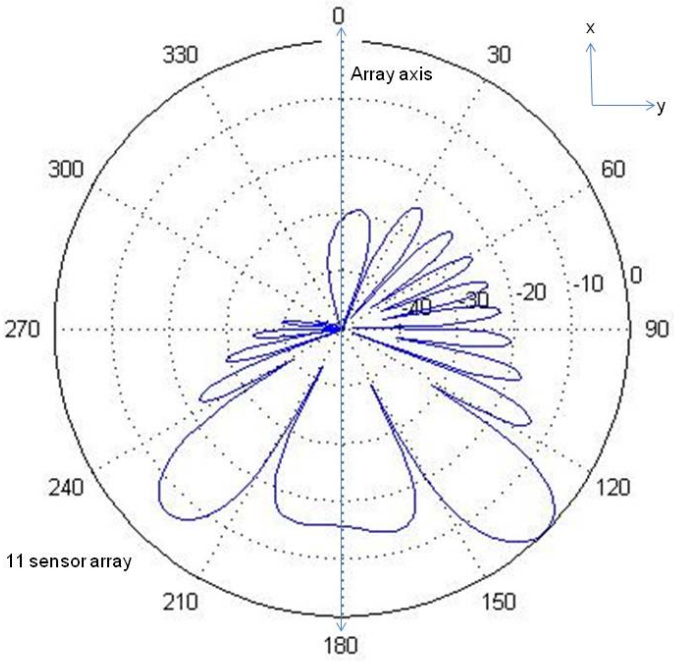

Figure 12 Cardioid Beampattern for a Vector Sensor Array Due to Incoming Signal at 135°

Studies have been completed comparing vector sensor arrays to pressure sensor arrays. These studies used linear cardioid processing for the vector sensor arrays and customary linear processing for the pressure sensor arrays. Results have shown that the directivity index[3] can be as much as 5 dB higher when comparing 10 sensor linear arrays (Cray & Nuttall, 2001). These results have encouraged further studies of vector sensor arrays.

[3] Directivity Index = 10 log(Directivity Factor).

c. Vector Sensor Array—Nonlinear Processing

The improvement shown in Figure 12 is encouraging; however, additional rejection of sound in the ambiguous direction may be possible. Consider a beamformer described by the equation

$$B_{dynnull}^{(pv)}(\theta_s,\phi_s) = \left| \sum_n b_n^{'(pv)}(\theta_s,\phi_s) e^{-ikx_n \cos\phi_s} \right|^2 \tag{9}$$

where $b_n^{'(pv)}(\theta_s,\phi_s) = (w_{xn}v_{xn} + w_{yn}v_{yn} + w_{zn}v_{zn} + w_{pn}^{'}v_{pn})$ with special weighting.

Note the weighting is the same as for cardioid processing for the velocity sensors; however, the weighting for the pressure sensor ($w_{pn}^{'}$) becomes, Aw_n, where $A = -\cos(2\phi_s)$. This new nonlinear (but not data adaptive) processing technique dynamically produces a null in the ambiguous direction (225°, Figure 13); consequently, it will be referred to as dynamic null processing here. Unfortunately, this technique also produces a substantial back lobe (in the vicinity of 0°).

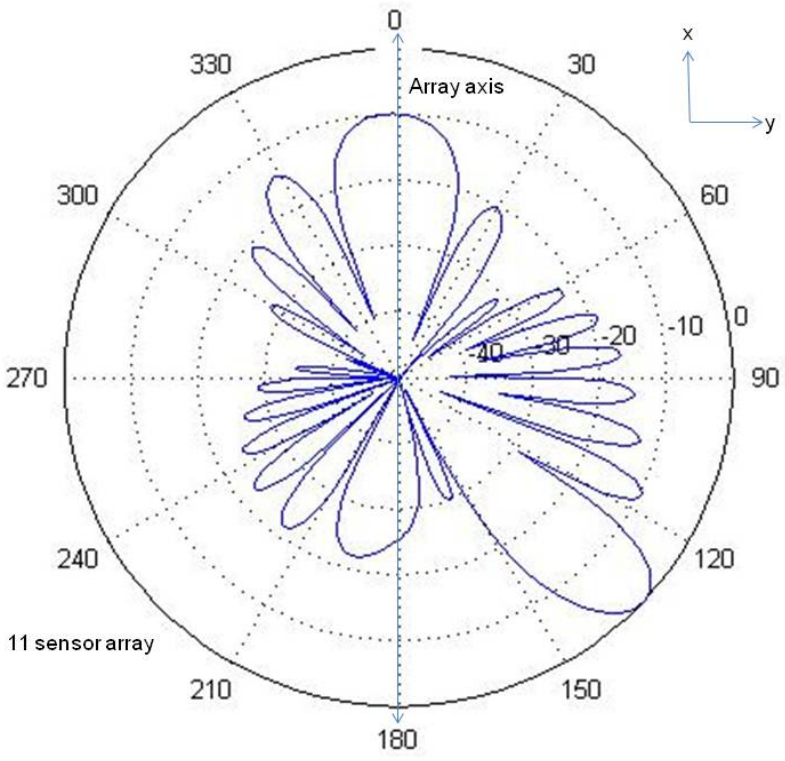

Figure 13 Dynamic Null Beampattern for a Vector Sensor Array Due to Incoming Signal at 135°

An alternative nonlinear technique is to incorporate some of the benefits of both cardioid processing (Figure 12) and dynamic null processing (Figure 13) by combining them in the equation

$$B_{cardynull}^{(pv)}(\theta_s,\phi_s) = \left|\sum_n b_n^{(pv)}(\theta_s,\phi_s)e^{-ikx_n\cos\phi_s}\right| \times \left|\sum_n b_n^{'(pv)}(\theta_s,\phi_s)e^{-ikx_n\cos\phi_s}\right| \qquad (10)$$

where $b_n^{(pv)}(\theta_s,\phi_s)$ has the standard weighting described in the cardioid section and, $b_n^{'(pv)}(\theta_s,\phi_s)$ has the special weighting described in the dynamic null section.

Since this processor combines cardioid processing with dynamic null processing, it will be referred to as cardynull (pronounced *card-e-null)* processing here.

Beampatterns for cardynull processing with signals broadside and at 135° are shown in Figure 14 and Figure 15. Figure 15 illustrates the promising aspects of cardynull processing with a large lobe in the signal direction (135°), a null in the ambiguous direction (225°) and a much smaller back lobe than the dynamic null processing technique (compare to Figure 13).

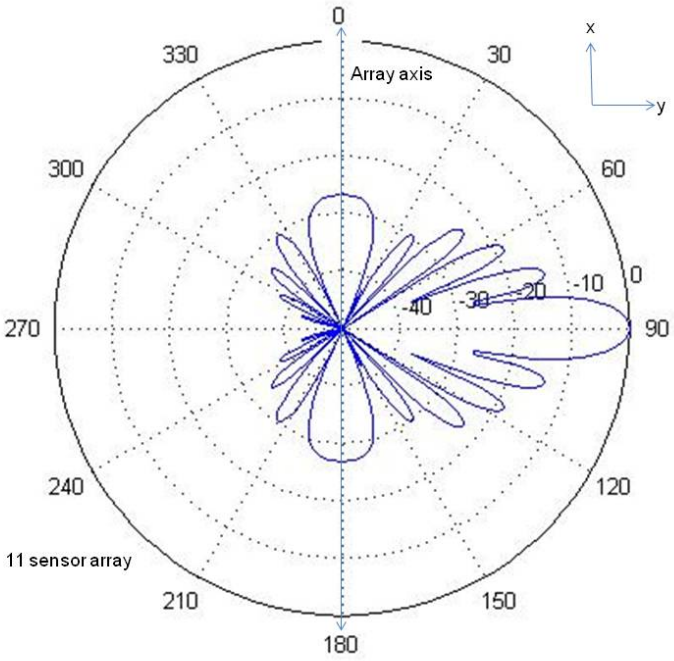

Figure 14 Cardynull Beampattern for a Vector Sensor Array Due to Incoming Signal at Broadside

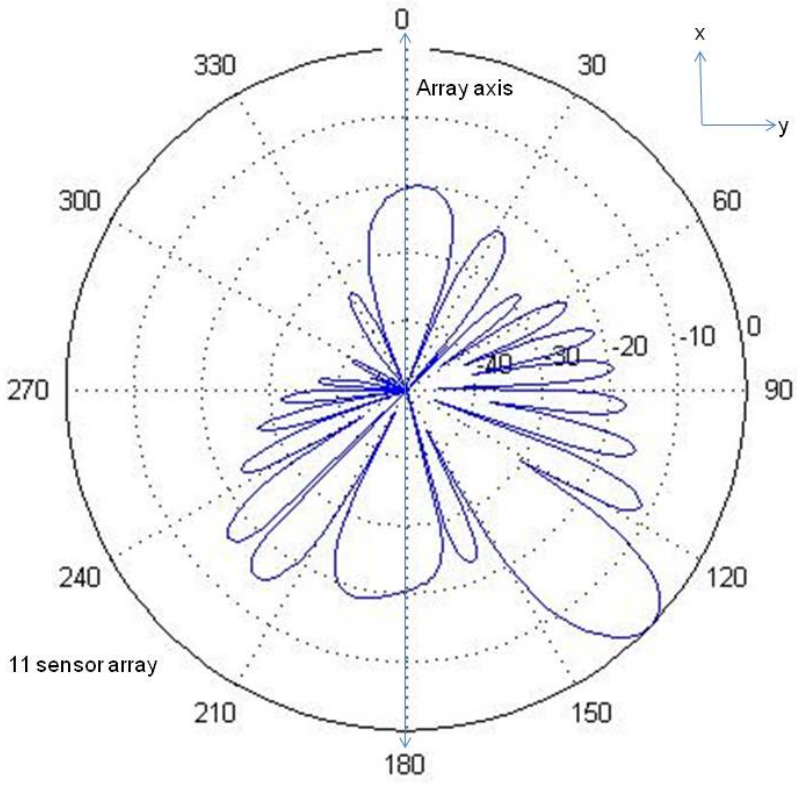

Figure 15 Cardynull Beampattern for a Vector Sensor Array Due to an Incoming Signal at 135°

B. PROBABILITY OF DETECTION

1. Probability Density Functions

Probability theory is often used to predict the aggregate behavior of large numbers of random variables. From probability theory, the central limit theorem states that the sum of a large number of random variables[4] will approximate a normal distribution. The normal or Gaussian distribution is given by the following equation and is displayed in Figure 16.

$$f(\chi) = \frac{1}{\sqrt{2\pi\sigma^2}} e - \frac{(\chi - \mu)^2}{2\sigma^2} \tag{11}$$

[4] Under conditions that are quite common in practical applications.

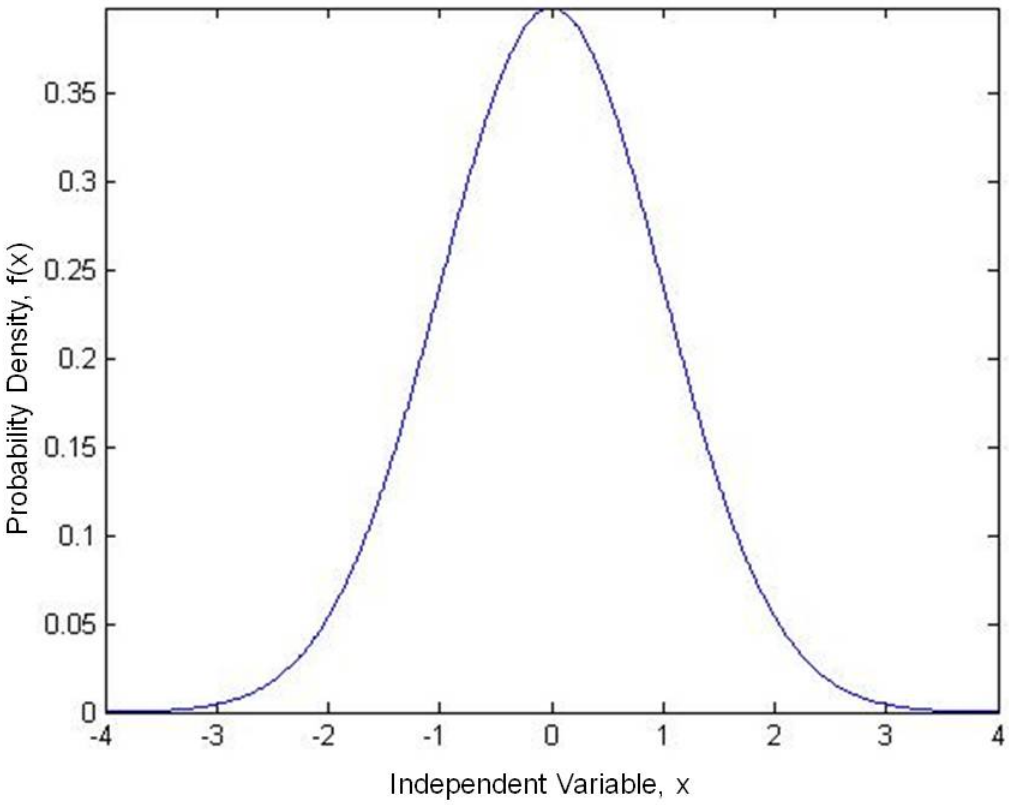

Figure 16 Probability Density Function for $\mu = 0, \sigma^2 = 1$

a. Noise Alone

The general probability density function like the one shown in Figure 16 can be used to characterize noise caused by the sum of a large number of random processes provided the independent variable ‘x’ in Figure 16 is replaced with the pressure or velocity amplitude as shown in Figure 17.

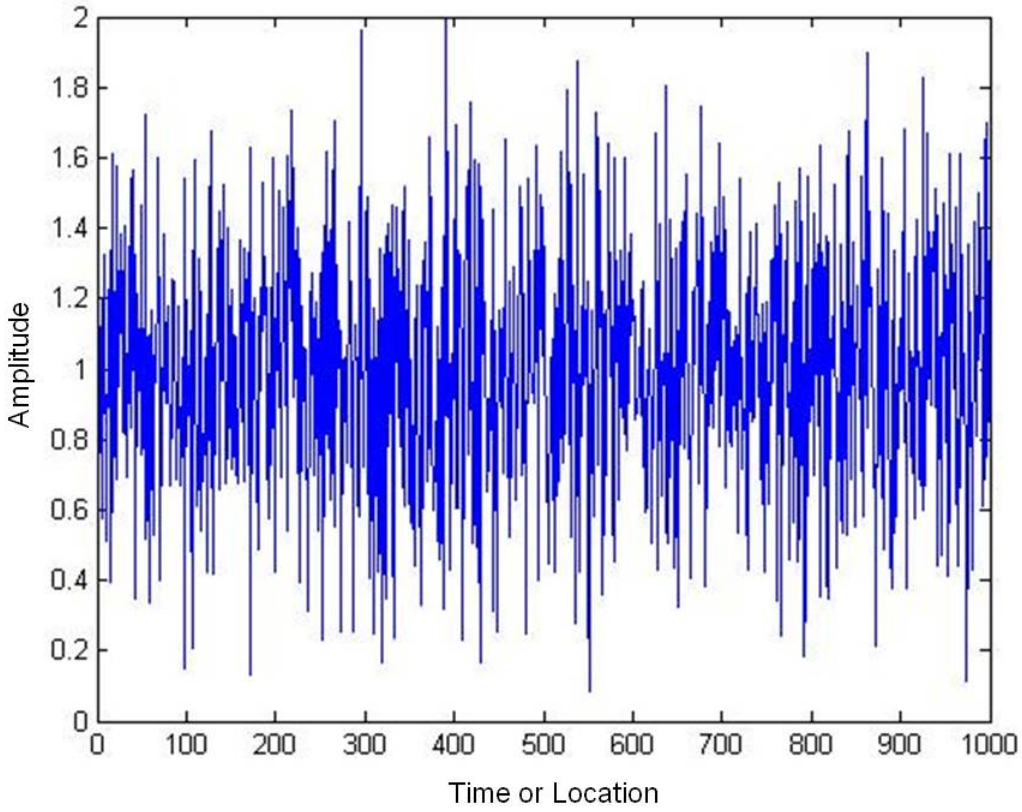

Figure 17 Example of Random Noise Amplitude

b. Signal and Noise

When a signal of interest (i.e., a target of interest) is added to noise the sum often has an amplitude similar to the noise alone as can be seen by comparing Figure 17 and Figure 18. To distinguish between a signal with noise (S&N) and noise alone, probability density functions for both can be plotted as shown in Figure 19. Even in cases where the signal and noise have similar amplitudes, the signal can be distinguished from the noise by spatial filtering (i.e., beamforming) to enhance localized sources, such as the signal of interest.

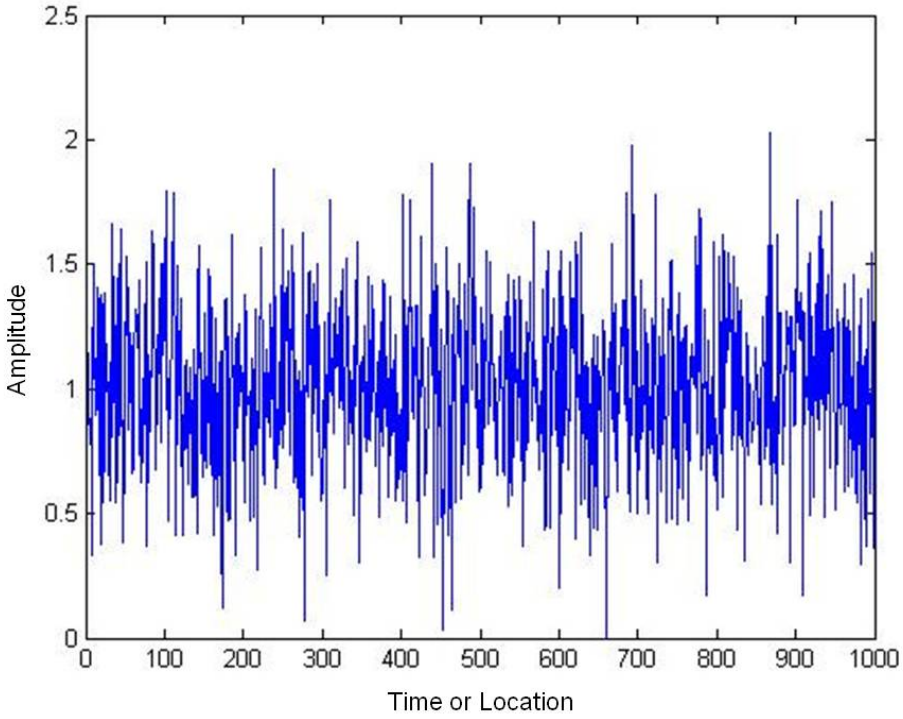

Figure 18 Example of S&N Amplitude

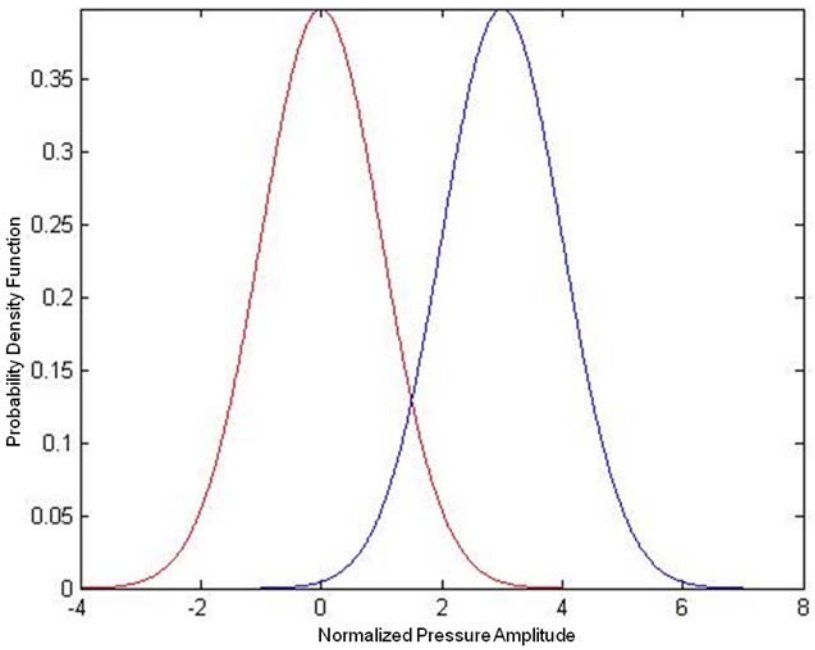

Figure 19 Probability Density Functions for S&N (blue) and Noise (red)

c. Target Detection in Noise

Target detection in noise is accomplished by introducing a threshold as shown by the black line in Figure 20. Everything above the threshold (i.e., to the right of the black line) is assumed to be the desired signal and everything below the threshold (i.e., to the left of the black line) is assumed to be noise. This obviously leads to some misclassifications since some signal waves with low amplitudes will be classified as noise while some noise waves with high amplitudes will be classified as the signal.

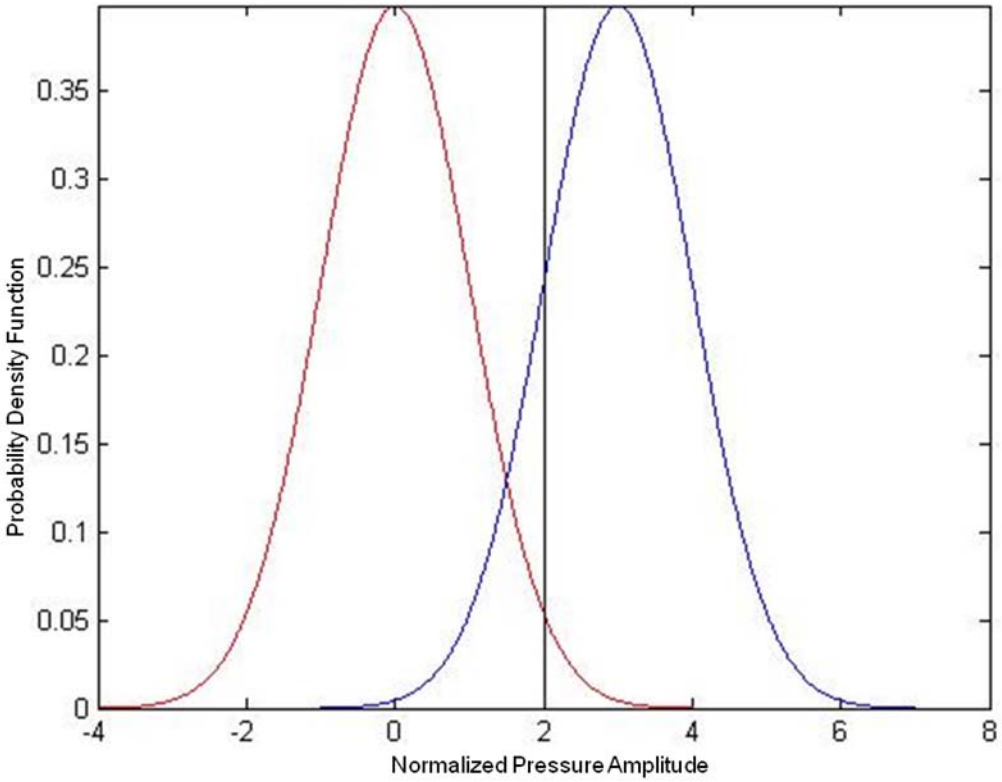

Figure 20 Signal and Noise with Detection Threshold (black)

2. Receiver Operating Characteristic (ROC) Curves

In Figure 20, the detection threshold can be adjusted to obtain various detection rates for both the signal and the noise. As the detection threshold is increased the probability of false detections (i.e., the false alarm probability) will decrease; however, also the probability of detecting the true signal will also decrease. Receiver Operating Characteristic (ROC) curves were developed (Peterson, 1953) to illustrate the effect of changing the detection threshold for a given population of signal and noise. A sample ROC curve is shown in Figure 21.

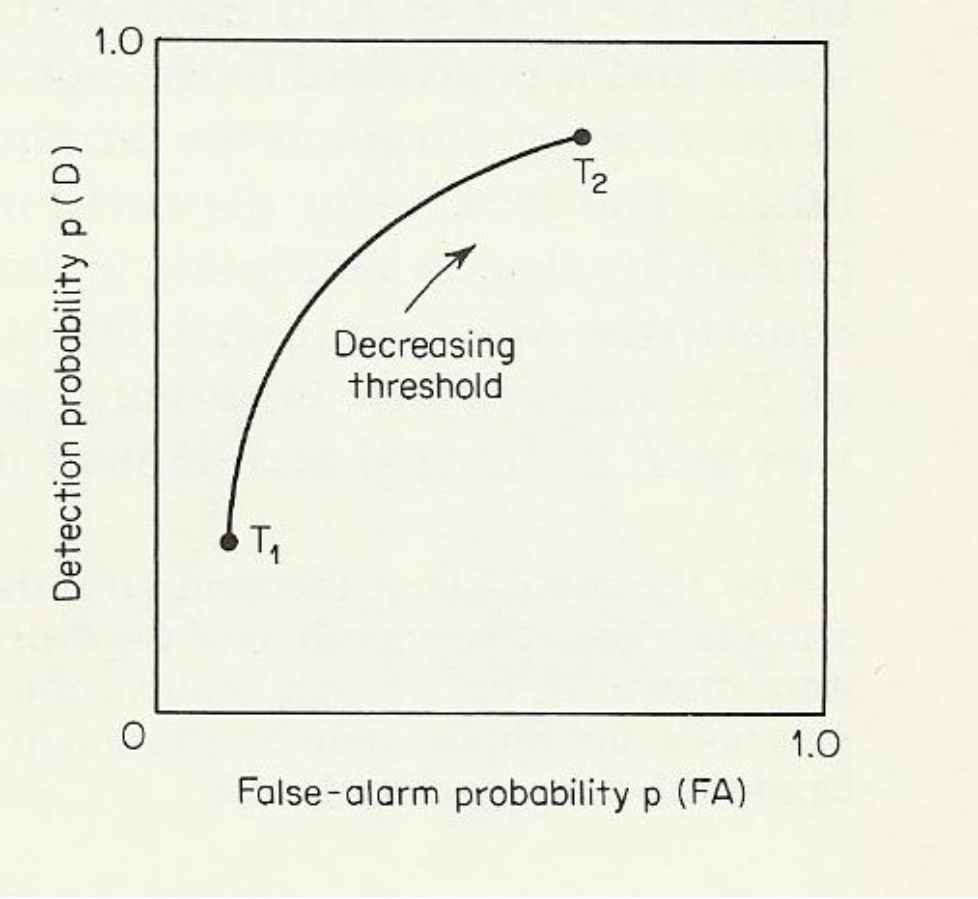

Figure 21 Sample ROC curve (From: Urick, 1983, p. 381)

An important characteristic of the ROC curves is that they show the effectiveness of a particular detection process. This is illustrated in Figure 22 for various values of detection index "d," where $d = \frac{\left(\mu_{S+N} - \mu_N\right)^2}{\sigma^2}$. As the two means (i.e., μ_{S+N} and μ_N) become separated, d increases and the ROC curve bows more toward the upper left of the graph. Increased "bowing" toward this upper left indicates more effective classification while the straight line between the lower left and the upper right (i.e., between (0,0) and (1,1)) indicates ineffective classification.

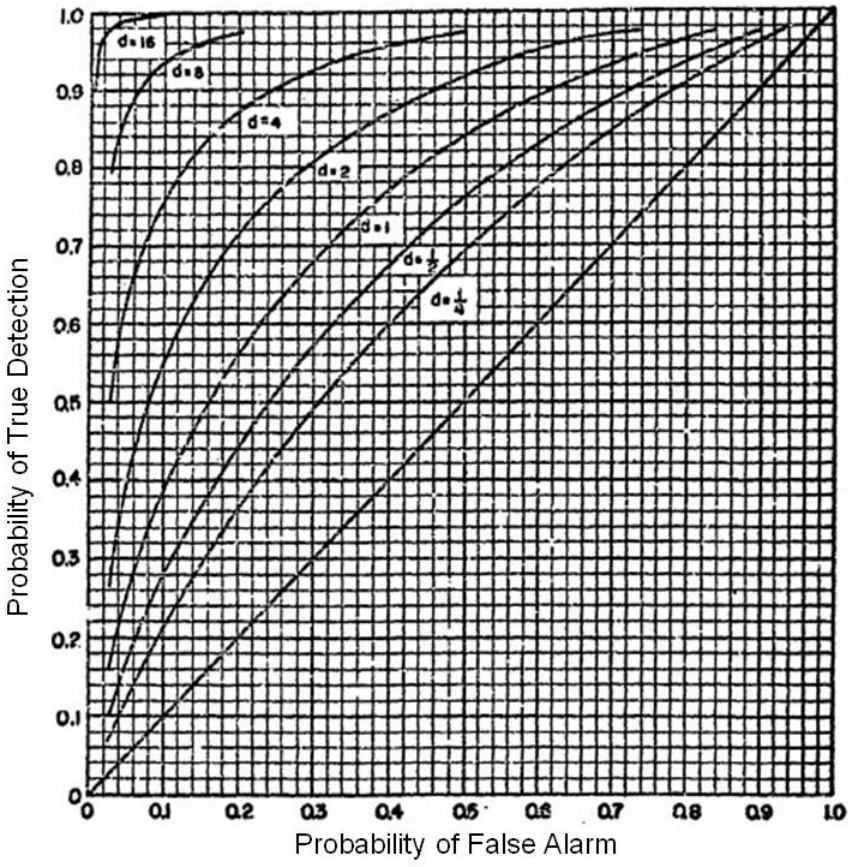

Figure 22 ROC curves for various values of "d" (From: Peterson, 1953)

III. COMPUTER PROGRAM

A. DESCRIPTION OF COMPUTATIONS

The main purpose of the computer code discussed in this thesis is to predict the effectiveness of various sensor arrays and processing techniques in simulated conditions. For undersea sound, these conditions imply that both a signal and noise will be present. Consequently, the effective classification of a signal in noise is considered and the results are displayed using Receiver Operating Characteristic (ROC) curves.

1. Coordinate System

The computer code described in this thesis uses the coordinate system shown in Figure 1.

2. Generation of Pressure/Velocity Waves

All pressure waves and velocity waves considered here are continuous waves (CW) with constant maximum amplitude and a random phase that remains consistent with $e^{-i\omega t}\left(\omega = 2\pi f\right)$. More complicated waves, such as frequency modulated (FM) waves were not considered. Use of CW-only signals allowed each of them to be represented simply by a complex number of the form a + ib (where a and b are real numbers).

The program user can specify the number of pressure waves. This number is used for both the signal and noise. For example, if the user selects 1000 pressure waves, the signal will be represented by 1000 complex numbers. Likewise, the noise will also be represented by 1000 complex numbers. The distribution of the maximum amplitudes is selectable by the user.

Velocity waves are formed based on the corresponding pressure waves. The amplitude of the velocity wave was determined from the formula $a_v = \frac{a_p}{\rho c}$,

where

a_v = amplitude of the velocity wave,
a_p = amplitude of the pressure wave,
ρ = density of water, and
c = speed of sound.

3. Arrangement of Signal and Noise Sources

The signal is a point source, so all of the waves that make up the signal are collocated at a point. For simplicity, the signal is located in the x-y plane. Consequently, for the analysis considered in this thesis, the signal is always located at a roll angle, θ, of zero. The primary program provides an illustration of the location of the signal, as shown in Figure 23. This illustration shows the angle of the signal from the array. It does not accurately display the distance of the signal from the array. The signal is assumed to be in the far-field of the sensors, and thus interacts with the array as a plane wave.

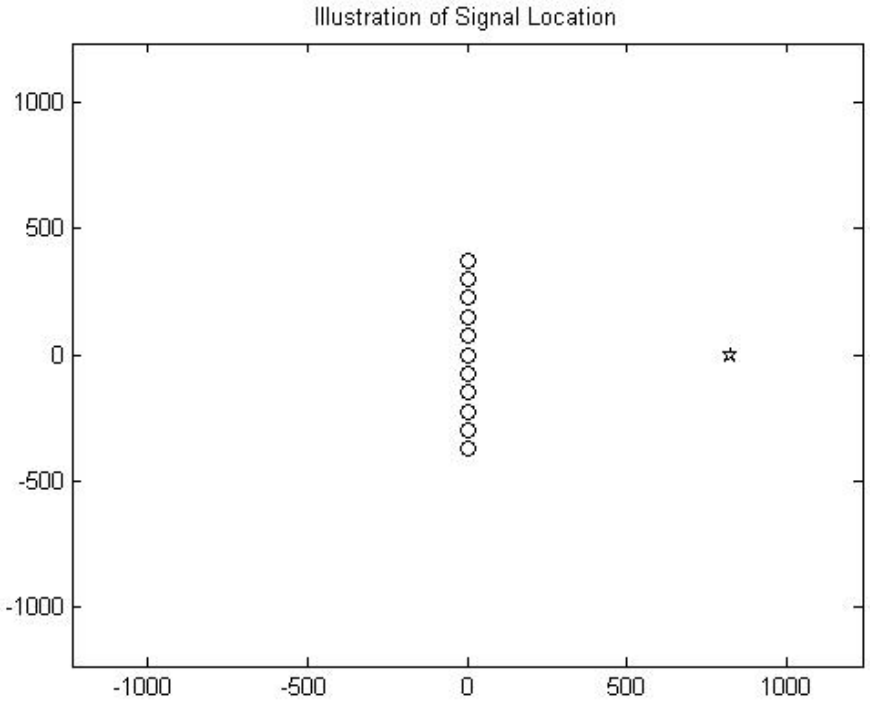

Figure 23 Sample Illustration of Signal Location

The noise is a distributed source, so the (ϕ,θ) location of each noise wave is randomly selected so that the noise waves cover a three-dimensional sphere around the array. An illustration of this arrangement is shown in Figure 24.

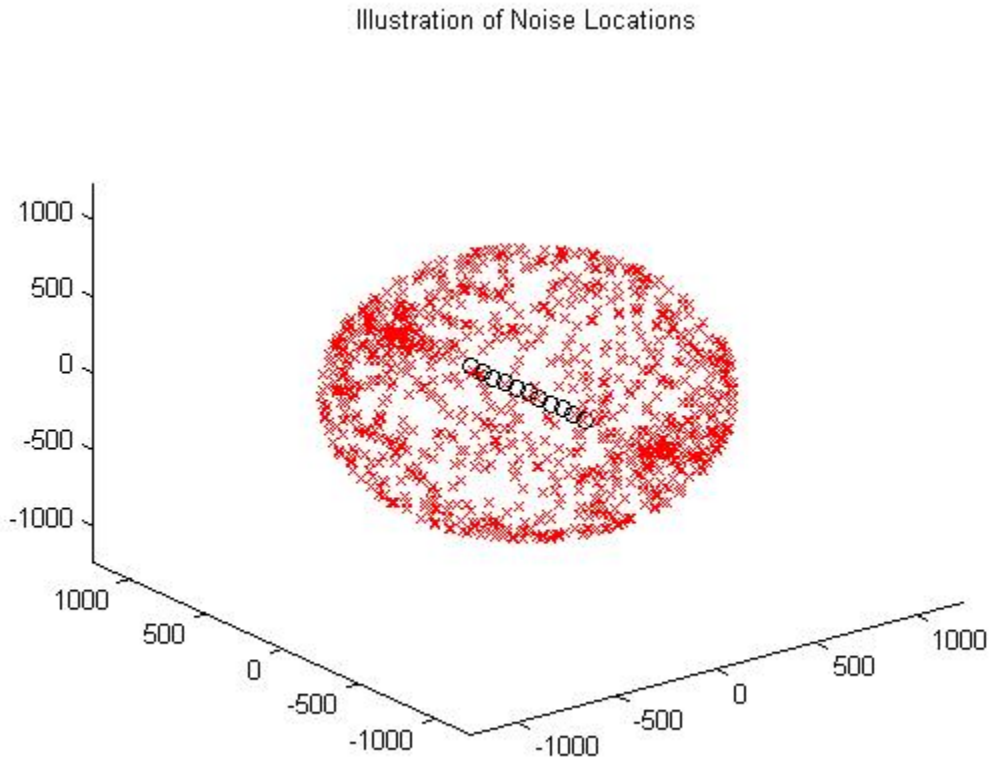

Figure 24 Sample Illustration of Noise Location

4. Pressure Wave Amplitude Distributions

The pressure waves may have a uniform distribution or a normal distribution. For the uniform distribution, the amplitudes are assigned within a range (i.e., $a = \mu \pm r$) such that all values within the range are equally probable. For the normal distribution, the amplitudes are assigned such that a normal or Gaussian distribution is obtained. For both signal waves and noise waves, the phases are uniformly distributed between 0 and 2π.

5. Adjustments for SNR; Calculation of Measured SNR

The user can specify the desired signal to noise ratio (SNR) in decibels (dB). The amplitude of each signal wave (pressure and velocity) is then adjusted by multiplying by the factor $\sqrt{10^{(SNR/10)}}$.

SNR is defined as (Urick, 1983)

$$SNR = \frac{\overline{S^2}}{\overline{N^2}} = \frac{\overline{(s_1 + s_2 + ...s_n)^2}}{\overline{(n_1 + n_2 + ...n_n)^2}}. \qquad (12)$$

This formula is used to calculate the measured SNR after all of the signal waves and noise waves are established. The validity of these equations may be evident by considering a few cases:

Example 1: When SNR = 0 dB, the multiplying factor above becomes 1. Then the average signal wave 's' is equal to the average noise wave 'n' and the SNR becomes 1, which is 0 dB.

Example 2: When SNR = 3 dB, the multiplying factor above becomes $\sqrt{2}$. Then the average signal wave power 's^2' is twice the average noises wave power 'n^2' and the SNR becomes 2, which is 3 dB (from the equation $SNR = 10\log_{10}(2) = 3dB$).

6. Monte Carlo Computations

A Monte Carlo simulation was used to simulate the variability of the signal and noise in the undersea environment. Each simulation involved a specified number of signal waves and noise waves, as discussed previously, and the response of sensors was also simulated and summed to produce beampatterns. While the patterns varied considerably, the summed result was normalized and a predictable average beampattern resulted as described in the next section.

Monte Carlo simulation was very useful for producing the statistical data required to generate the probability of target detection and the probability of false alarms. These values were used to generate ROC curves that illustrate the effectiveness of various processing techniques.

7. Beamforming with Signal and Noise

a. Pressure Sensor Array

Figure 6 provided the response of a pressure sensor array to a broadside signal alone. In undersea acoustics, however, the noise is always present, and responses to the more practical situations of noise alone and signal with noise are of interest. Figure 25 provides the response of the same array to noise originating from all directions. It was obtained by randomly distributing pressure waves throughout a three-dimensional space defined by the ranges $\phi = 0°$ to 360° and $\theta =$ -90° to 90° (see Figure 24) then normalizing and summing the resulting beampatterns. The response is more uniform than the signal only case shown in Figure 6, but the array geometry does distort the pattern from the omni-directional pattern (Figure 2) of a single sensor. The result shown in Figure 25 is reasonable since steering toward endfire causes the primary lobe to widen causing a greater response from noise waves in that region.

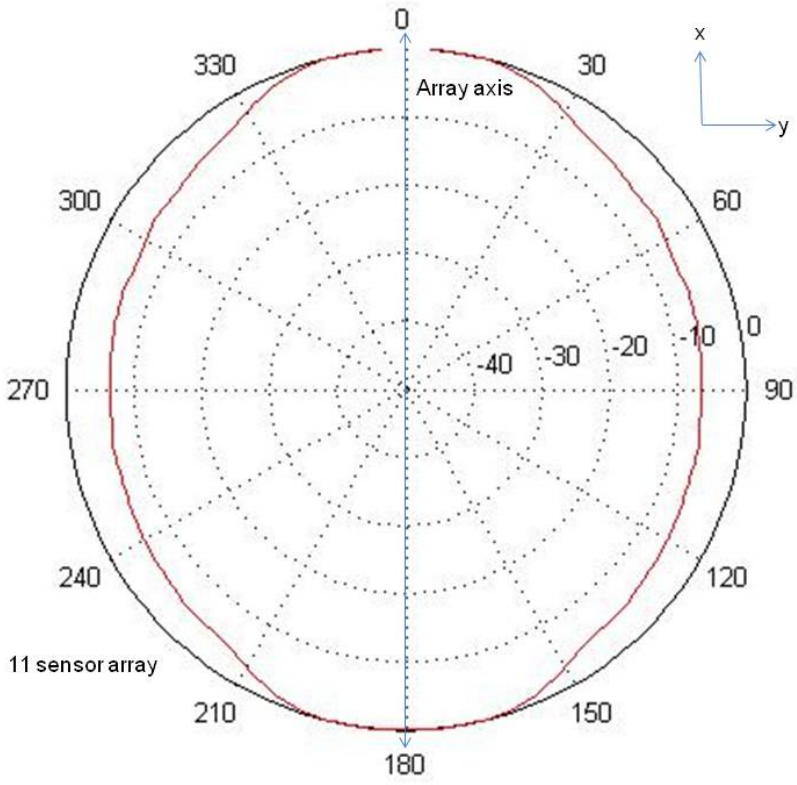

Figure 25 Pressure Array Response to Noise

When signal and noise are combined and averaged, the results show some similarities to the signal only (Figure 6) and noise only (Figure 25) plots shown above. Figure 26 shows such a combination when the signal power is equal to the noise power (i.e., when SNR = 0 dB). Array Gain is defined in the following equation (Urick, 1983) for a perfectly coherent signal with perfectly incoherent noise. Consequently, the 10-decibel gain shown in Figure 26 agrees with theory.

$$AG = 10\log(N) \qquad (13)$$

where

AG = Array Gain, and
N = number of sensors in the array.

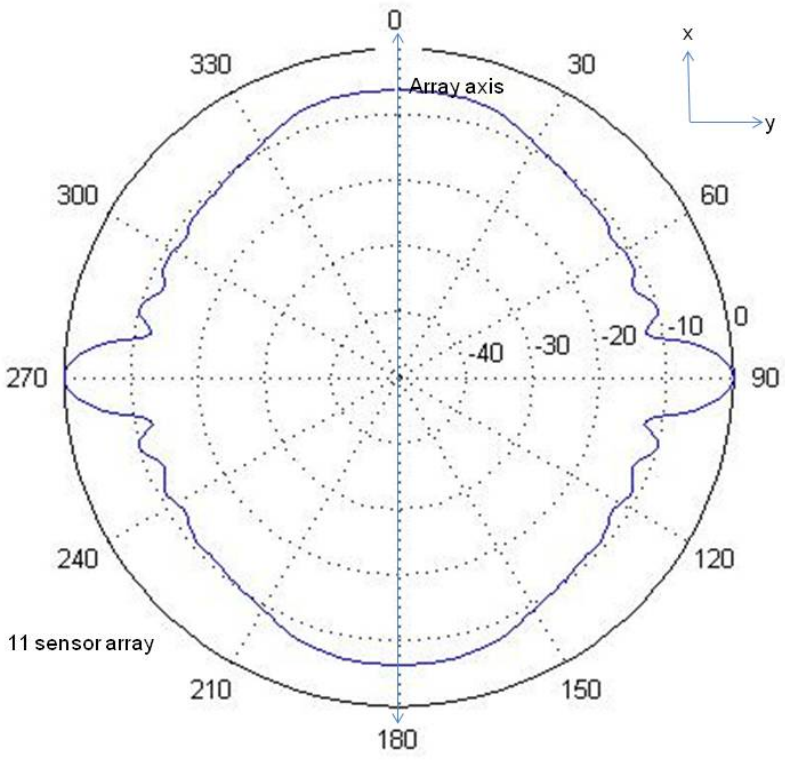

Figure 26 Pressure Array Response to Signal and Noise (SNR = 0 dB)

b. Cardioid Array

Figure 11 provided the linear cardioid response of a vector sensor array to a broadside signal alone. Figure 27 provides a similar plot for both the signal and noise.

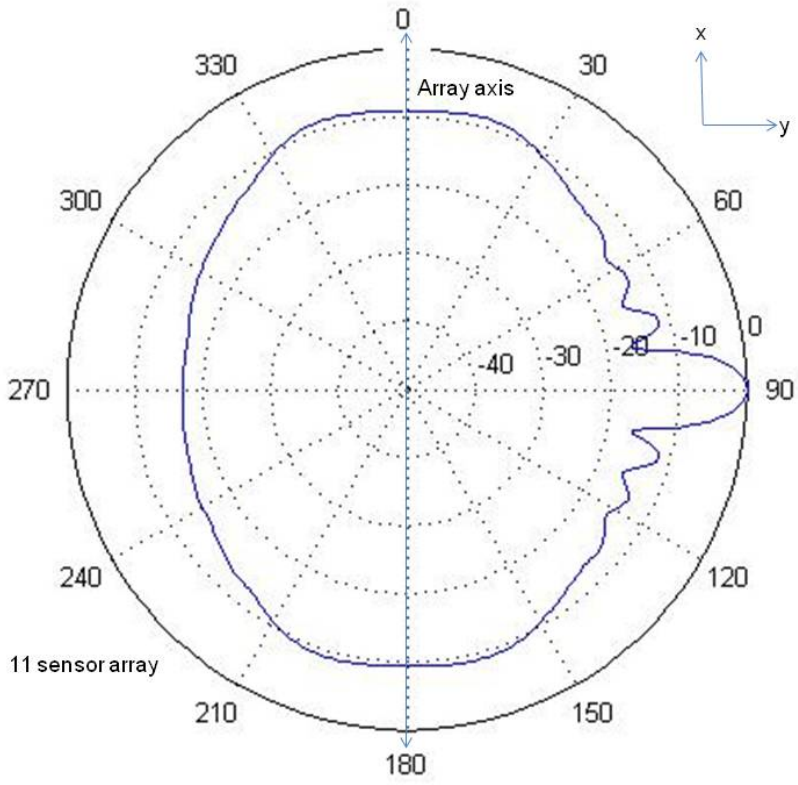

Figure 27 Cardioid Response to Signal and Noise (SNR = 0 dB)

c. Cardynull Array

Figure 14 provided the nonlinear cardynull response of a vector sensor array to a broadside signal alone. Figure 28 provides a similar plot for both the signal and noise.

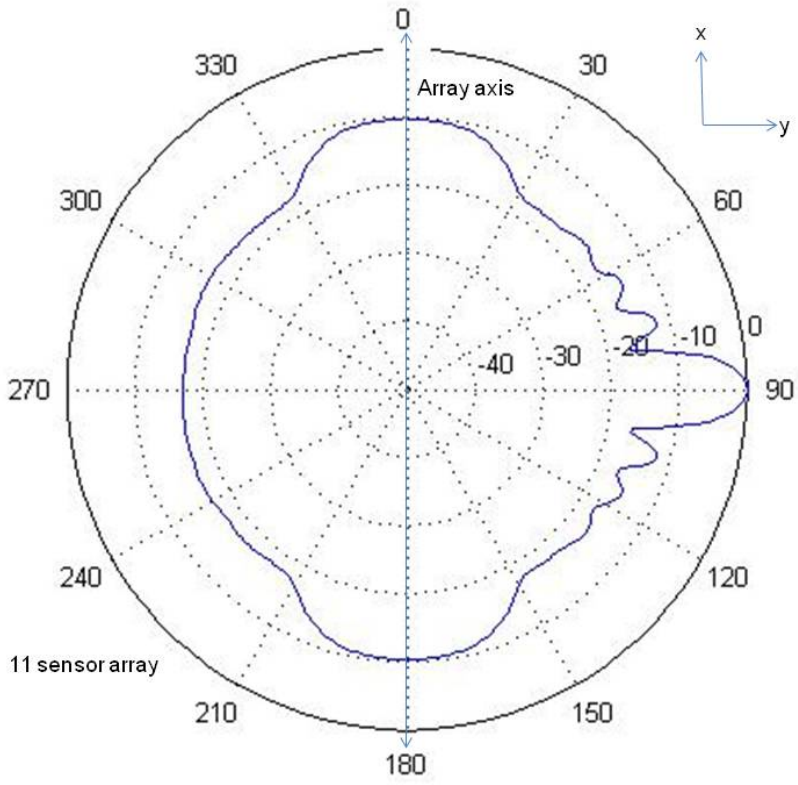

Figure 28 Cardynull Response to Signal and Noise (SNR = 0 dB)

8. Detection

This section considers the detection of a signal in noise. The technique used in this thesis was to classify all responses above a given threshold as a hypothesized positive; then, since the actual location of the signal was known, these positives were further classified as true positives or false positives. This process was repeated for various detection thresholds. The process can be visualized by referring to Figure 26, Figure 27, and Figure 28; however, these figures actually are the averaged responses for each processing technique. To obtain the detection probability data required, this process was executed for each Monte Carlo run. For each case, the hypothesized positive was classified as a true positive if it was located within a small tolerance (e.g., ± 5° or ± 15°) of the known location of the signal.

a Pressure-Only Arrays—Accommodation for Ambiguity

As discussed earlier, pressure-only sensor arrays provide a symmetric response around the array axis that results in a signal cone of ambiguity. This can be observed in Figure 26. As a consequence, a classifier that considers only hypothetical positives in the vicinity of the signal to be true positives will always have false positives resulting from the ambiguous direction. Since the positives that result from the ambiguous direction are produced specifically from the signal strength, it is necessary to consider these as true positives to create appropriate detection statistics. Consequently, for pressure-only sensor arrays, if the response was classified as a hypothetical positive (by exceeding the specified detection threshold); it was classified as a true positive if it was located either within a small tolerance of the signal location or within a small tolerance of the ambiguous direction.

b. Cardioid Array

The cardioid array presented the simplest detection situation. After the pressure and particle velocity components for each sensor were linearly combined these responses were summed with the other sensors in the array. Hypothetical positives were established for each response above the specified detection threshold. Finally, the true positives were simply classified as those hypothetical positives within a small tolerance of the signal direction.

c. Cardynull Array—Accommodation for Nonlinearity

The cardynull array uses the same vector sensors as the cardioid array described above. Consequently, processing is similar. However, due to the nonlinear aspect of this processing technique, the maximum response does not always coincide with the signal location. For example, Figure 29 shows the beampattern for a vector sensor array with the cardynull processing when the signal is located at 170°. In this case, the maximum response is at 159°.

Due to the difference between the signal location and the maximum response, a short program was written to calculate the expected location of the maximum response for arrays of various lengths and signal locations. This program produced a table that provides the expected maximum response as a function of array length and signal location for arrays between 2–150 sensors and for sensor locations between 10°–170° and between 190°–350°. Values at endfire (171°–189° and 351°–9°) were not calculated since cardynull processing becomes impractical when the signal and ambiguous directions become aligned.

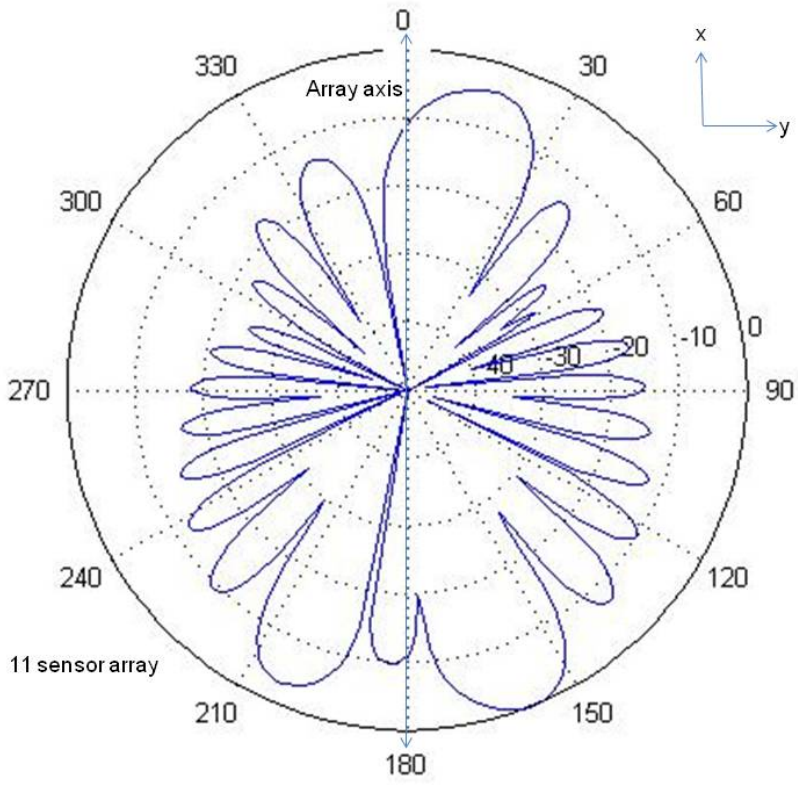

Figure 29 Cardynull Beampattern for Signal-Only when Near Endfire

9. ROC Curve Generation

A final important step in the process was the generation of the Receiver Operating Characteristic (ROC) curves. For each Monte Carlo run, it was

determined if at least one true positive existed or if at least one false positive existed. If these conditions were satisfied, the summation of true positives or true negatives was increased by one. When all of the Monte Carlo runs were completed, these summations were divided by the total number of Monte Carlo runs. The process was repeated for various detection thresholds and for each of the processing techniques under consideration (i.e., pressure, cardioids, and cardynull). The results included paired lists of true positive rates and false positive rates for each processing technique as established by the series of detection levels. When each paired list of true positive rate versus false positive is plotted, ROC curves, such as Figure 30 are obtained.

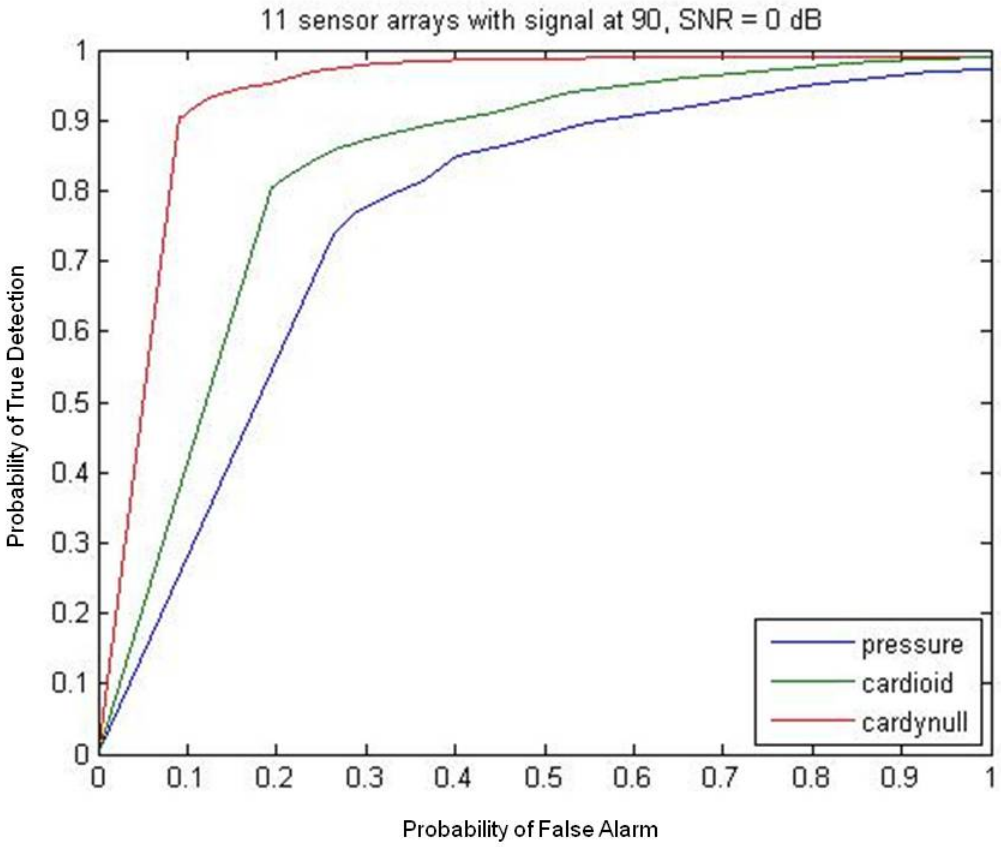

Figure 30 ROC curves for 11 Sensor Arrays with Signal Broadside[5]

[5] All ROC curves between broadside and 45° from broadside use a detection angle of ± 5°.

B. PROGRAM ORGANIZATION

Computer programs and supporting functions were created using MATLAB® (Release, 2010b).[6] Table 1 provides the hierarchy of the primary program, ROCmaker2, and the supporting functions. ROCmaker2, is listed in Appendix A. The supporting functions are divided into levels based on how they are called. Functions called directly by ROCmaker2 are labeled as Level 1 functions below and included in Appendix B. Some Level 1 functions call other functions that are labeled as Level 2 and included in Appendix C. Some Level 2 functions in turn call Level 3 functions that are included in Appendix D.

[6] MATLAB® is a registered product of Mathworks®, Natick, Massachusetts.

Table 1 Program Hierarchy

Main ROCmaker2	Level 1	Level 2	Level 3
	makeMatrix_n2		
	arraySensorLocation1		
	showArrayArrangement1		
	showSignalLoation2		
	showArrayArrangement2		
	showNoiseLocation3		
	beamPatternFunction5		
		generatePressAmpPhs2	
		generateVelAmpPhs2	
		calculateWavenumber1	
		beamSteer5	
			calculateWavenumberSteering4
			defineWeightingFactorsPress1
			defineWeightingFactors3
			calculateA
			combineSensors1
	plotBeampatterns4		
		plotScaling5	
		polar6	
		signalExpectation3	
			equation2131e
			cardioidBeamPatt2
			cardynullBeamPatt2
		noiseExpectation1	
			equation2131e
		rateCalc2	
			aboveTheshold2
			edgeDetect3
			detectRegion1
			slopeDetermine1
			locationMax2
			sonarBlip4

1. Main Program

The main program “ROCmaker2” is listed in Appendix A. The primary purpose of the main program is to allow for user input data and to organize the inputs and outputs to each of the Level 1 functions. Critical acoustics calculations are not included in this main program; instead, they are allocated to supporting functions.

Important inputs to “ROCmaker2” include the desired signal to noise ratios, the number of sensors in the array and the location of the signal. As a minimum, the program will output a ROC curve for each specified SNR with a curve for each sensor processing technique (see Figure 30). The program also can display average beampatterns for each processing technique (see Figure 26, Figure 27, and Figure 28) and other preliminary data that will be discussed in the function sections below.

2. Functions

The functions are divided into levels as discussed above and listed in Appendices B, C, and D. The purpose of each function is discussed below.

a. Level 1

makeMatrix_n2: Creates a one-dimensional matrix with a length equal to the number of sensors in the array (i.e., <nx1>). The number of sensors can be either even or odd. The function output provides the distance of each sensor from the origin in increments of $\lambda / 2$.

arraySensorLocation1: Using the output of “makeMatrix_n2,” this function specifies the location of each sensor in three-dimensional space (x,y,z). All arrays in this thesis are linear and located along the x-axis; therefore, the *y* and *z* positions for each sensor is zero. Since the wavelength (λ) is provided in millimeters, the *x* location of each sensor is also provided in millimeters.

showArrayArrangement1: Provides a two-dimensional display of the sensor array.

showSignalLocation2: Adds the signal location to the two-dimensional display.

showArrayArrangement2: Provides a three-dimensional display of the sensor array.

showNoiseLocation3: Adds the location of the noise sources to the three-dimensional display.

beamPatternFunction5: Creates and displays beampatterns for various processing techniques (i.e., pressure, cardioid, and cardynull).

plotBeampatterns4: Displays beampatterns for various processing techniques including signal only, noise only, and combined signal and noise.

rateCalc2: Calculates true positive and false positive rates for use in ROC curves. This function can provide limited optional displays of the locations of true positives and false positives.

b. Level 2

generatePressAmpPhs2: Creates the pressure wave amplitudes and phases. The user may specify that the pressure amplitudes are a uniform distribution or a normal distribution. If a uniform distribution is selected, each amplitude value between a maximum and a minimum are equally probable. If a normal distribution is selected the amplitudes will form a normal or Gaussian distribution. The phase for each pressure wave is a uniform distributed random number.

generateVelAmpPhs2: Creates the velocity wave amplitudes and phases consistent with the pressure waves. The velocity amplitude of each wave is determined by taking the pressure wave amplitude and dividing by density (ρ) and the speed of sound (*c*). The phase of the velocity wave is the same as the phase of the pressure wave.

calculateWavenumber1: Calculates the components of the wavenumber $(\text{i.e., } k_x, k_y, k_z)$ given the wavelength (λ), and the location of the signal $(\text{i.e., } \phi \text{ and } \theta)$. Since an incoming plane wave is assumed, the wavenumber components are typically negative.

beamSteer5: For an array of any length, this function steers the beam around the entire circle (0-360 degrees) and sums the result. It does this for any specified processing technique (i.e., pressure, cardioid, or cardynull).

plotScaling5: Transforms the data to a decibel format with a maximum of zero.

polar6: This function is a modified version of the MATLAB® standard function polar. It accepts the data transformed by "plotScaling5" and outputs beampatterns with a maximum of 0 dB and a minimum of -50 dB. It allows space for a title at the top of the plot.

signalExpectation3: This function provides optional output as controlled from the function "plotBeampatterns4." It determines the expected beampattern when only the signal is present. It can provide a beampattern for any of the processing techniques (i.e., pressure, cardioid, or cardynull).

noiseExpectation1: This function provides optional output as controlled from the function "plotBeampatterns4." It determines the expected beampattern when only noise is present. It only has been utilized for pressure-only sensor arrays.

aboveThreshold2: Determines if a received level is above or below the detection threshold. The results are stored in two variables: the first records a value of one if the received level is above the detection threshold, the second records the received level if it is above the detection threshold. Received levels below the detection threshold are set to zero.

edgeDetect3: Determines the location of the left and right edges of each detection regions.

detectRegion1: Groups each detection pair (i.e., left/right edge of the detection region) into a matrix. The output “detectPairs” matrix has a size of <# of detection regions, 2>.

slopeDetermine1: For values within a detection region, this function determines the slope of the receive level by comparing adjacent receive levels.

locationMax2: This function determines the approximate location of the maximum receive level.

sonarBlip4: This function determines if the maximum determined by “locationMax2” is likely a true positive or a false positive. It does this by comparing the maximum location to the expected signal location. If the maximum is within a tolerance (default $\pm 5^{\circ}$) of the expected signal location, it is assumed to be a true positive. For nonlinear cardynull processing, an adjustment is made in the expected signal maximum location. This adjustment is pre-calculated by the program “cardynullTablemaker” and the accompanying function “cardynullTablemakerFunc2” and placed in a table that is supplied to “sonarBlip4.”

c. Level 3

calculateWavenumberSteering4: Calculates the wavenumber for the steering vector.

defineWeightingFactorsPress1: Sets the weighting factors for the pressure-only sensor array. As a default, uniform weighting was used. Other weighting, (e.g., Hanning) could also be used.

defineWeightingFactors3: Sets the weighting factors for vector sensor arrays (cardioid or cardynull processing).

calculateA: This short function calculates "A" using the following formula: $A = -\cos(\phi_s)$.

combineSensors1: This function combines the contribution from each sensor in the array. The result is then squared (for pressure or cardioid processing) or combined with a dynamic null (for cardynull processing).

equation2131e: This function provides the beampattern predicted by using vanTrees equation 2.131.

cardioidBeamPatt2: This function provides the beampattern expected for cardioid processing as predicted by a program originally developed by K. B. Smith.

cardynullBeamPatt2: This function provides the beampattern expected for cardynull processing as predicted by a program originally developed by K. B. Smith.

3. Program and Function to Calculate Cardynull Maximum Response

The program "cardynullTablemaker.m" and the function "cardynullTablemakerFunc2.m" produce a matrix with the location of the expected maximum signal response as a function of signal location and number of sensors. The matrix is also stored as "cardTable.mat." The first column is the signal location in degrees (0°–359°). Columns 2–150 provide the expected maximum signal response location in degrees for signals located between 10°–170° and between 190°–350°. The values near endfire (i.e., 351°–9° and 171°–189°) are not accurate and the users are cautioned to not use these values.

The nonlinearity of the cardynull response becomes less significant as the number of sensors is increased. At 150 sensors, the location of the maximum signal response for cardynull processing becomes effectively equal to the location of the signal. Consequently, users wishing to simulate cardynull processing of sensor arrays larger than 150 sensors can neglect the cardynull nonlinearity and anticipate the maximum response from the signal direction.

IV. RESULTS

A. DETECTION PERFORMANCE

This section discusses results obtained using "ROCmaker2" to simulate the detection of a signal in noise. The simulations discussed below were completed using certain defaults that can be assumed to exist unless specifically stated otherwise. These defaults include:

- Signal represented by 1,000 pressure waves (and corresponding velocity waves) with normal random amplitude and uniform phase all collocated at a point in the array far field.
- Noise represented by 1,000 pressure waves (and corresponding velocity waves) with normal random amplitude and uniform phase with random locations uniformly distributed in three-dimensional space at points in the array far field.
- Array spacing equal to one half a wavelength (i.e., $\frac{\lambda}{2}$).
- Array containing 11 sensors
- Using 1000 Monte Carlo runs to generate detection statistics
- Using a SNR value of 0 dB
- The detection tolerance was ±5° for broadside to 45° from broadside. The detection tolerance was ±15° for 40° from endfire to endfire.

1. At Broadside for Various SNR Values

The response of pressure only array, a vector sensor array using cardioid processing, and a vector sensor array using cardynull processing to a broadside signal was already presented in Figure 30. These results for a signal level equal to the noise level illustrate the advantage of vector sensors compared to pressure sensors and the additional advantage that cardynull processing can achieve over the customary linear (cardioid) processing. The same processing improvements are noted when the signal power is doubled (+3 dB) or halved (-3 dB) compared to the noise level (see Figure 31 and Figure 32).

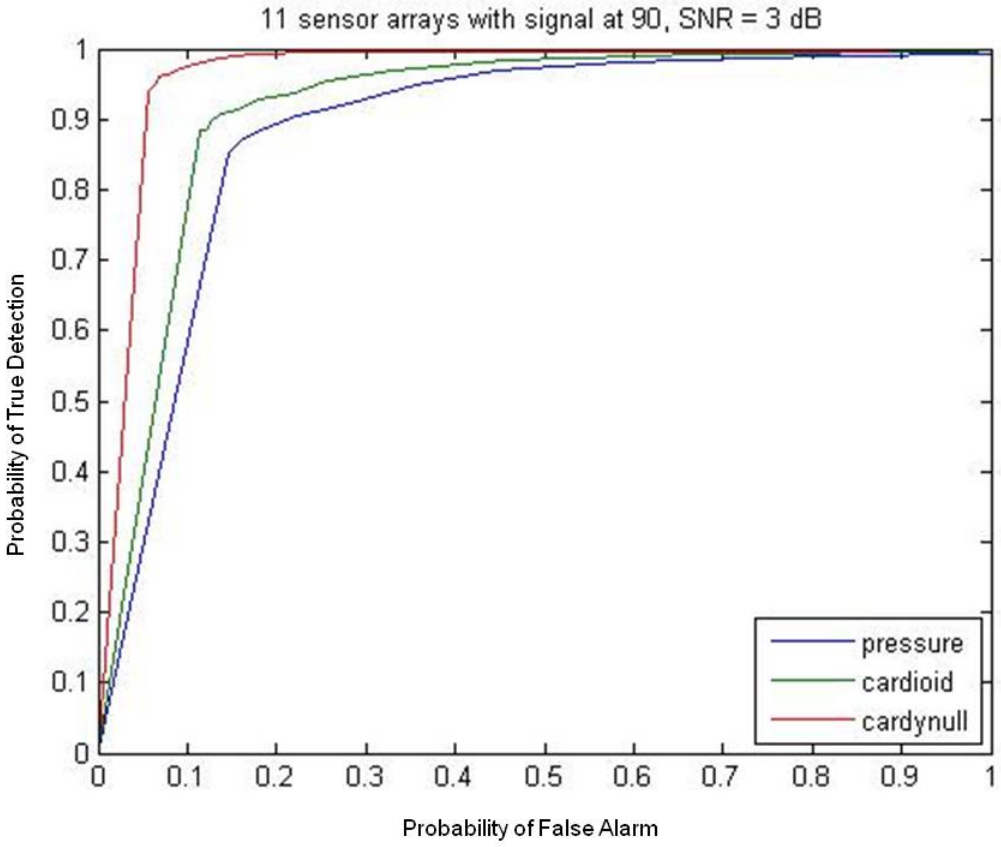

Figure 31 Comparison of Processing Technique for SNR = 3 dB

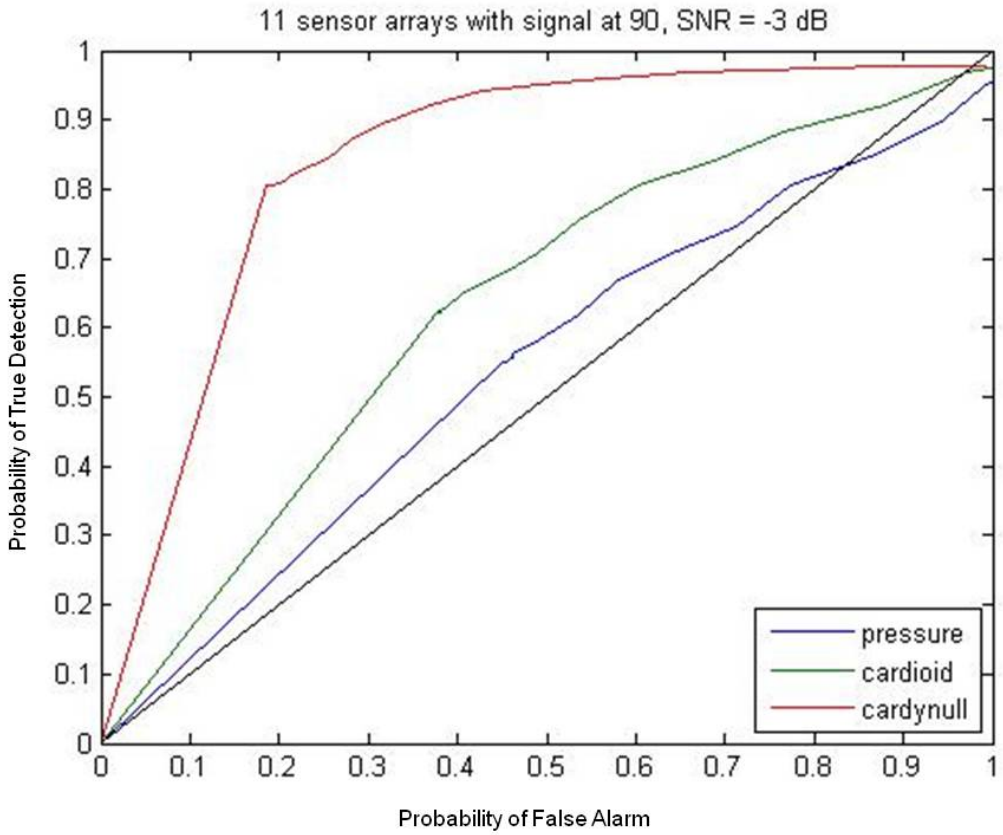

Figure 32 Comparison of Processing Technique for SNR = -3 dB

In Figure 32, the pressure sensor array becomes fairly ineffective. This is evident when the ROC curve for pressure is compared to the black line shown. The black line represents the ROC curve if we had no detection information and we simply flipped a coin to guess if the signal was present or not.

Considering nonlinear cardynull processing in comparison to linear cardioid processing for the same vector sensor array, we can see (Figure 33) that cardynull processing offers approximately a 3 dB improvement.

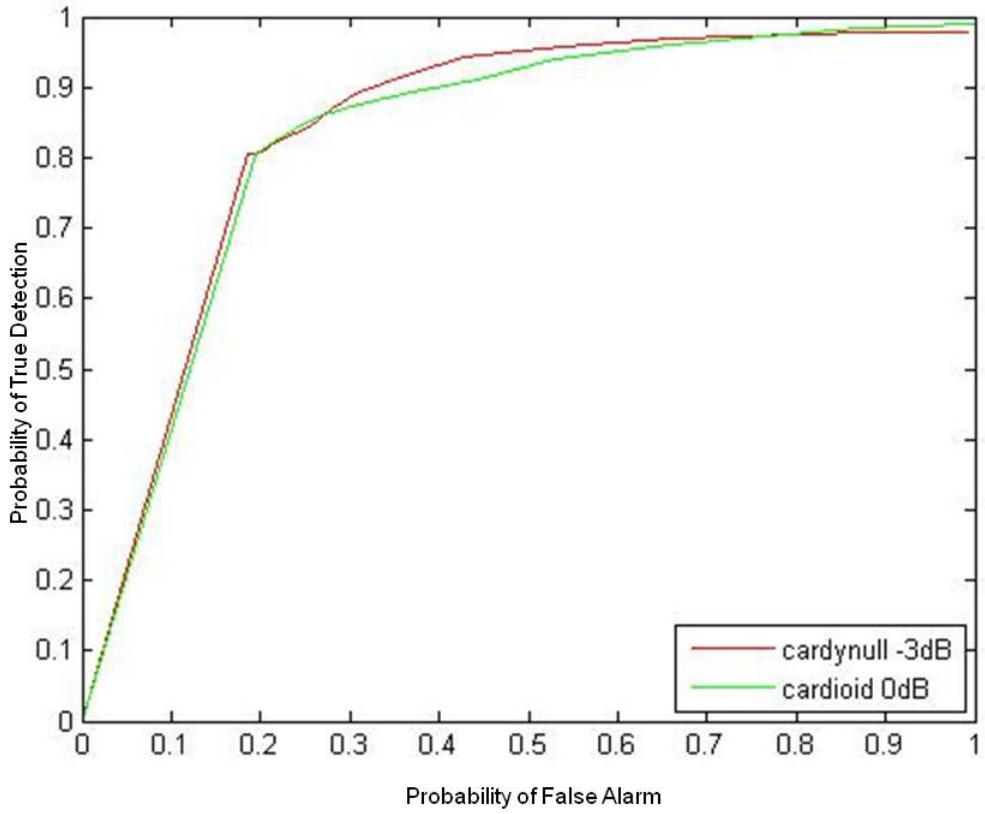

Figure 33 Cardynull Processing compared to Cardioid Processing

2. Away from Broadside and Endfire

Considering a signal location as much as 45 degrees from broadside, Figure 34 and Figure 35 show that the same trend remains; namely, nonlinear cardynull processing provides the best performance and cardioid processing provides better performance than pressure-only sensor arrays.

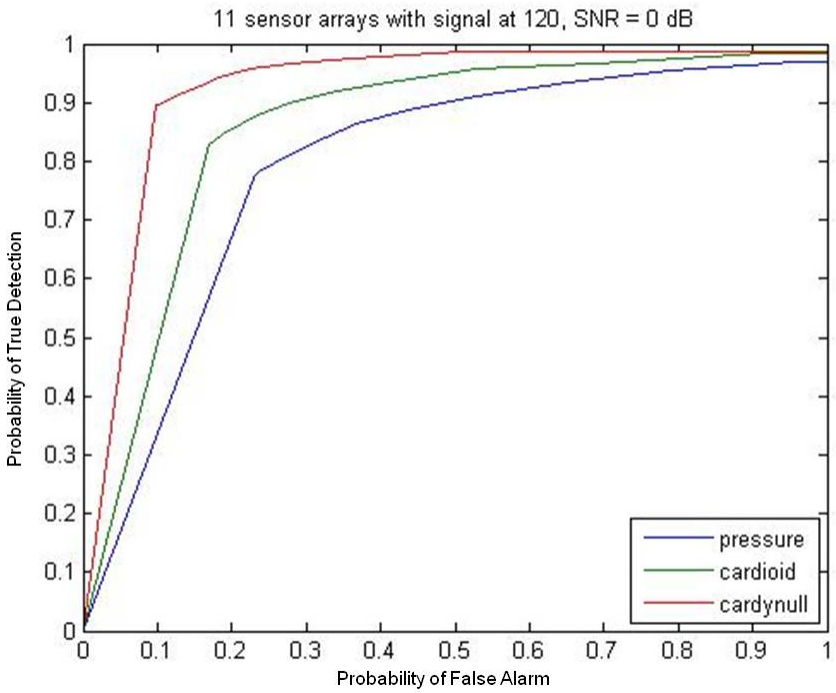

Figure 34 ROC Curves for Signal at 30 degrees from Broadside

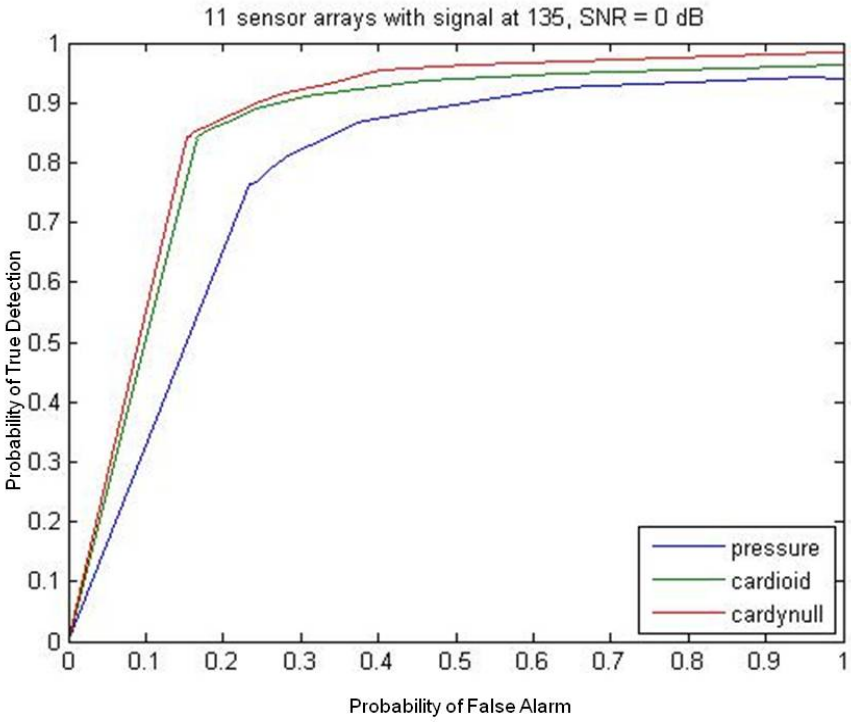

Figure 35 ROC Curves for Signal at 45 degrees from Broadside

3. Approaching Endfire

The cardynull processing technique becomes less effective than the other techniques as the signal approaches endfire (see Figure 36, Figure 37, and Figure 38). This could be due to how this processing technique reduces the response in the ambiguous direction or it could be due to the nonlinear aspect of the processing, which may amplify uncorrelated noise more distinctly as the processing becomes more nonlinear toward endfire.

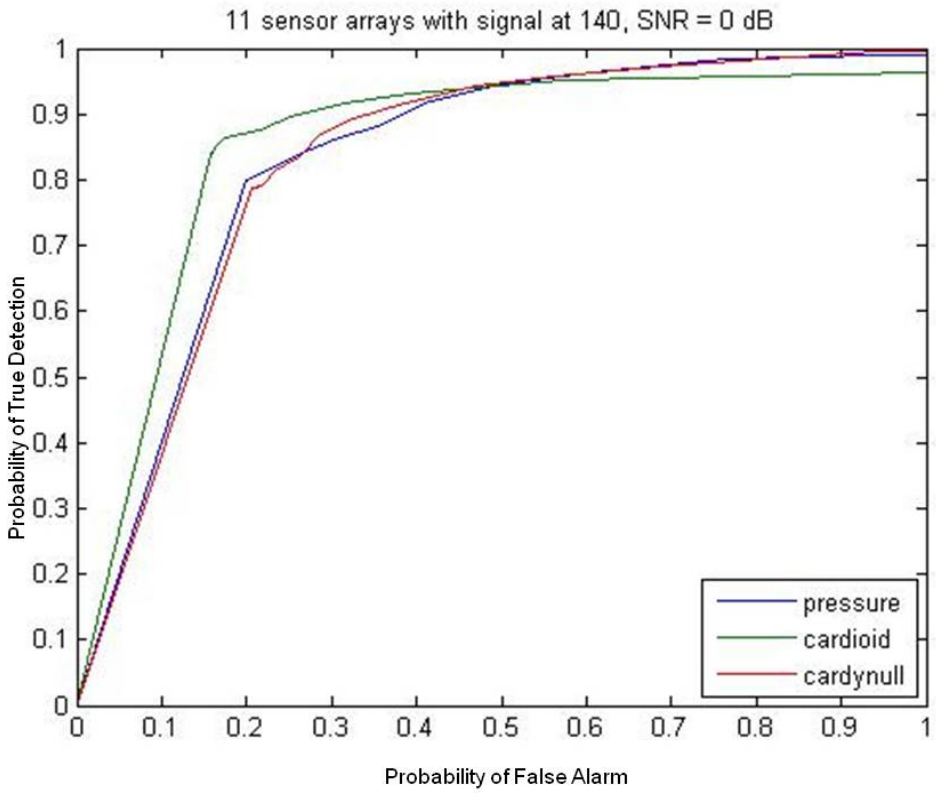

Figure 36 ROC Curves for Signal at 50 Degrees from Broadside[7]

[7] All ROC curves between 45° from broadside and endfire use a detection angle of ± 15°.

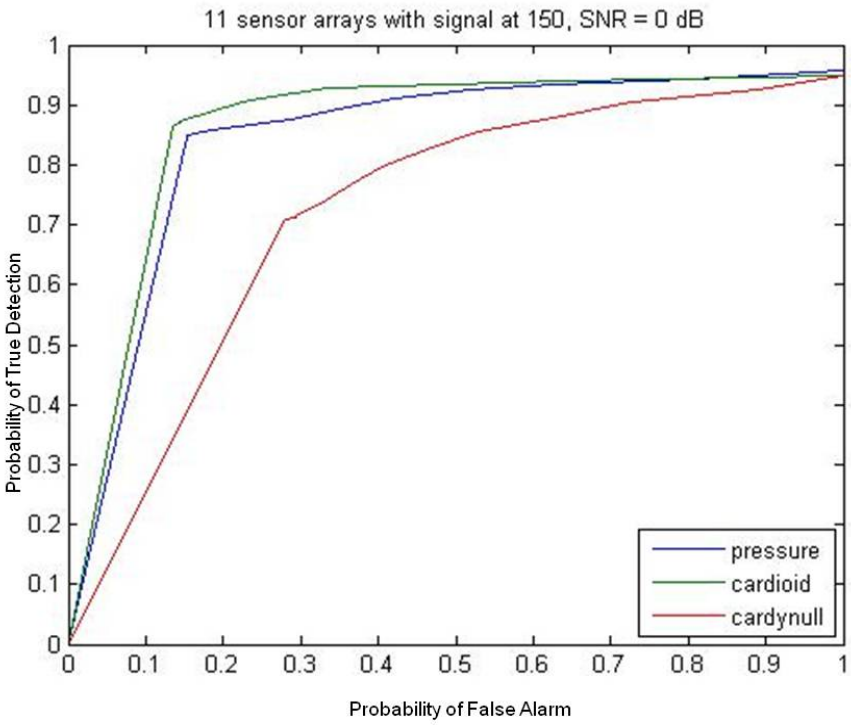

Figure 37 ROC Curves for Signal at 60 Degrees from Broadside

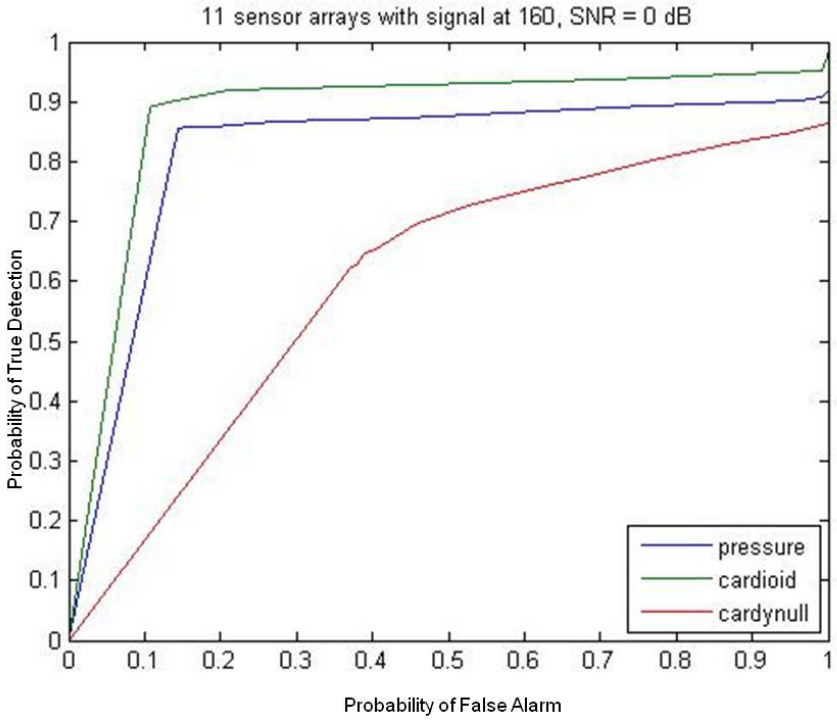

Figure 38 ROC Curves for Signal at 70 Degrees from Broadside

This relative degradation in detection performance for cardynull versus cardioid processing is not limited solely to short arrays. The trend still is evident in long arrays (for example, 100 sensors) as shown in Figure 39.

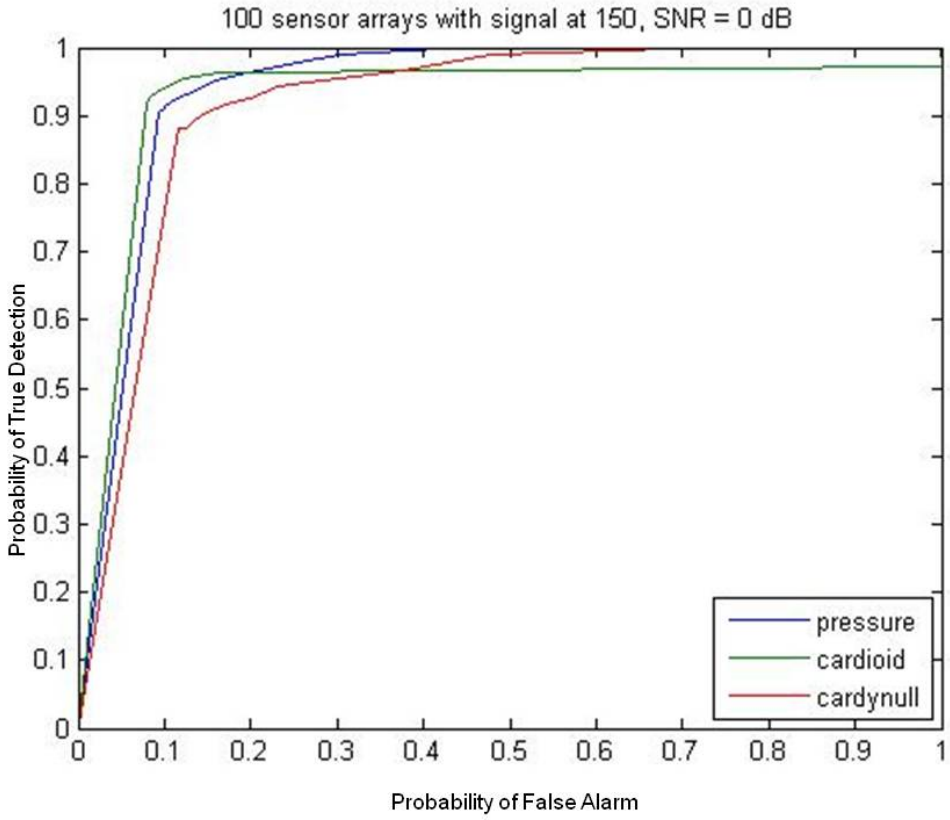

Figure 39 ROC Curves for 100 Sensor Arrays With Signal 60 Degrees from Broadside

B. EFFECT OF CHANGING NUMBER OF PRESSURE WAVES

Figure 40 provides a comparison of results for 10,000 pressure and velocity waves versus 1,000 pressure and velocity waves. This causes a small difference in the curves without changing the trends between the processing techniques.

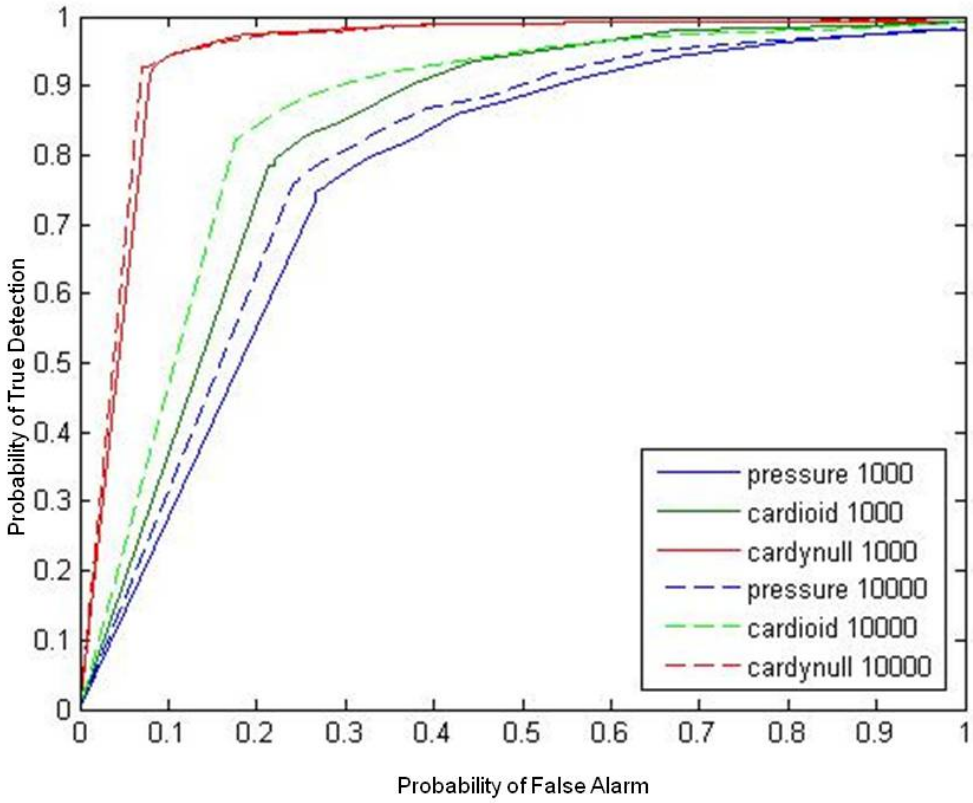

Figure 40 ROC Curve Comparison of Number of Pressure/Velocity Waves

C. EFFECT OF CHANGING NUMBER OF MONTE CARLO RUNS

Figure 41 shows the effect of increasing the number of Monte Carlo runs to 10,000. This plot shows this effect is minimal although we might expect that additional Monte Carlo sets with 1,000 runs would form a distribution around the Monte Carlo set with 10,000 runs.

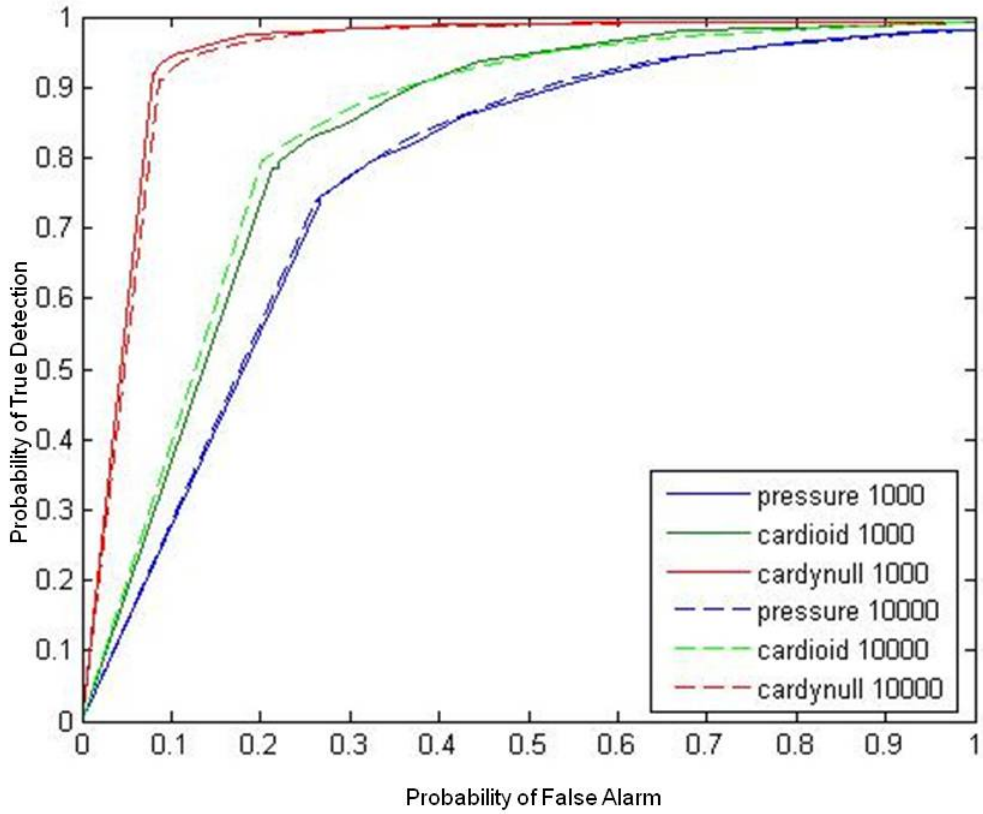

Figure 41 ROC Curve Comparison of Number of Monte Carlo Runs

D. EFFECT OF CHANGING AMPLITUDE DISTRIBUTION

"ROCmaker2" allows the user to choose a random uniform distribution or a random normal distribution for the pressure wave amplitude. The velocity wave amplitude is calculated from the pressure wave amplitude; therefore, the velocity waves will have the same distribution as specified for the pressure waves. Figure 42 shows a small difference between the amplitude distributions; however, the trends between the processing techniques remain the same.

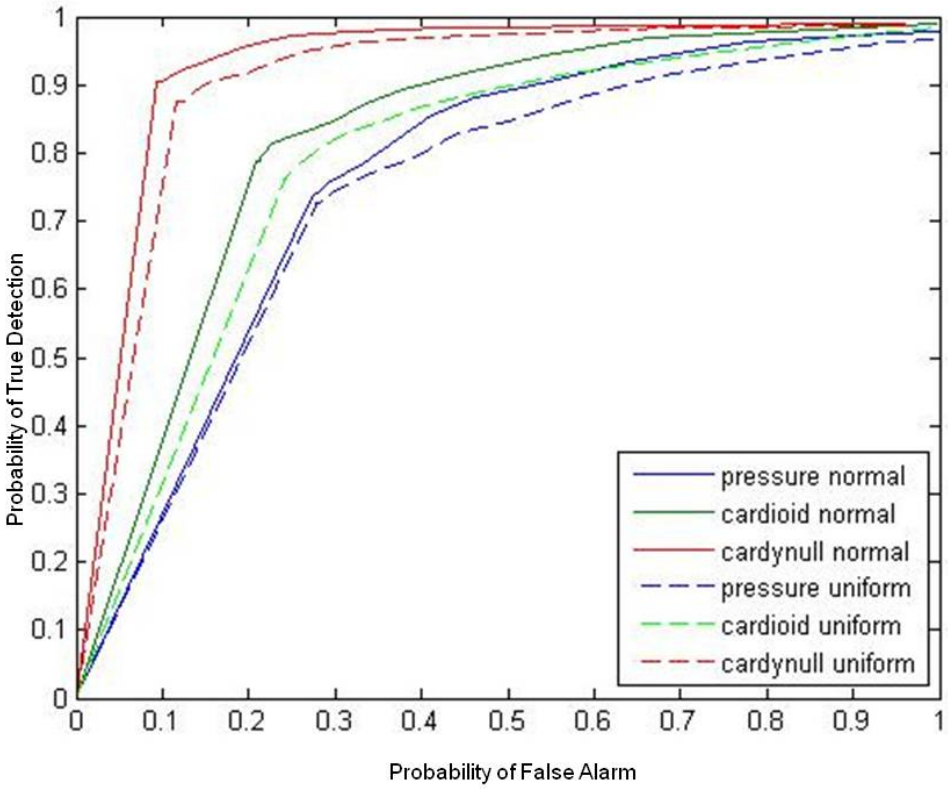

Figure 42 ROC Curve Comparison for Wave Amplitude Distributions

E. EFFECT OF ARRAY LENGTH

Many modern sonar arrays utilize large numbers of sensors to increase array directivity as shown in Figure 43, Figure 45, and Figure 47. This greatly improves detection capabilities as we might expect and as shown in Figure 44, Figure 46, and Figure 48.

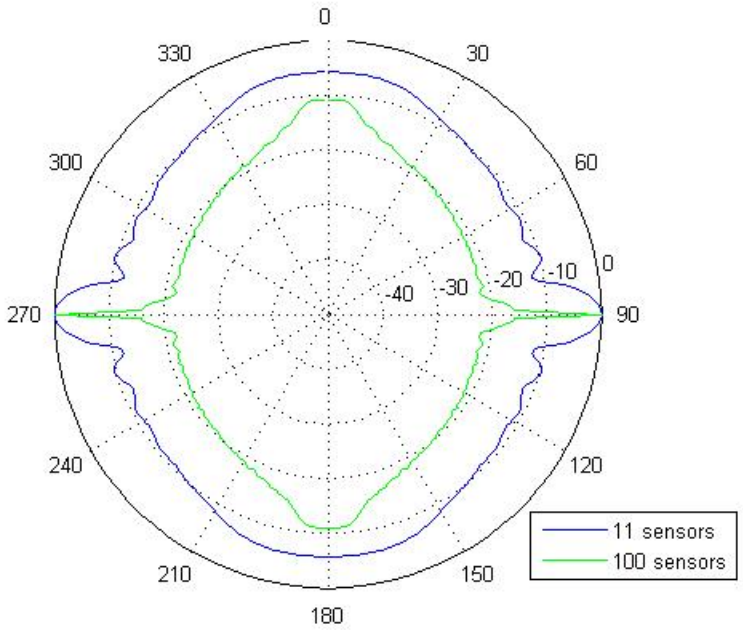

Figure 43 Beampattern Comparison for Pressure Arrays

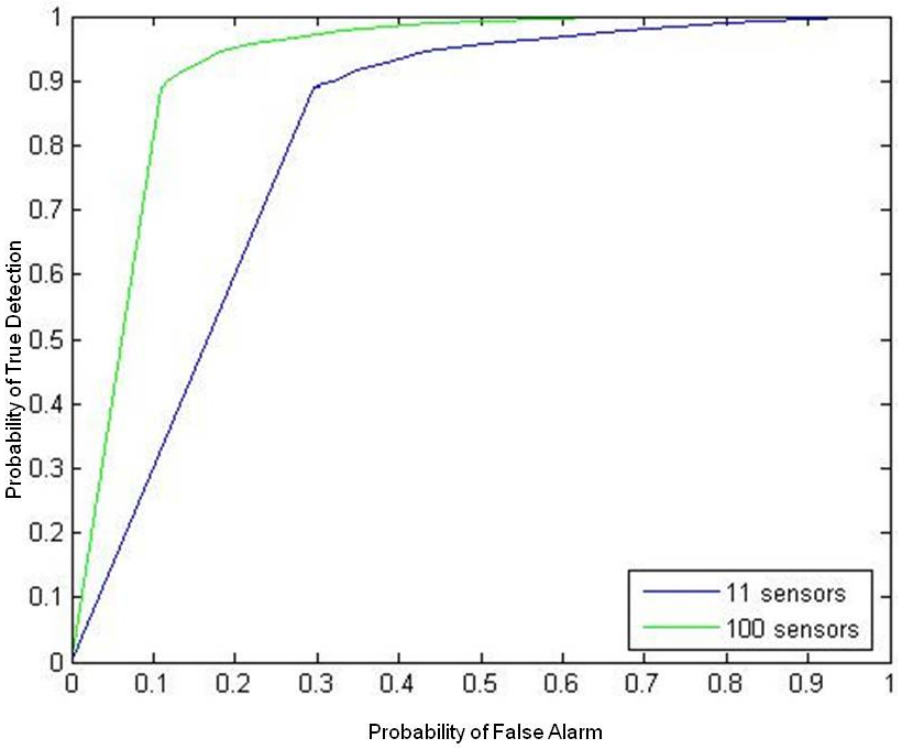

Figure 44 ROC Curve Comparison for Pressure Arrays

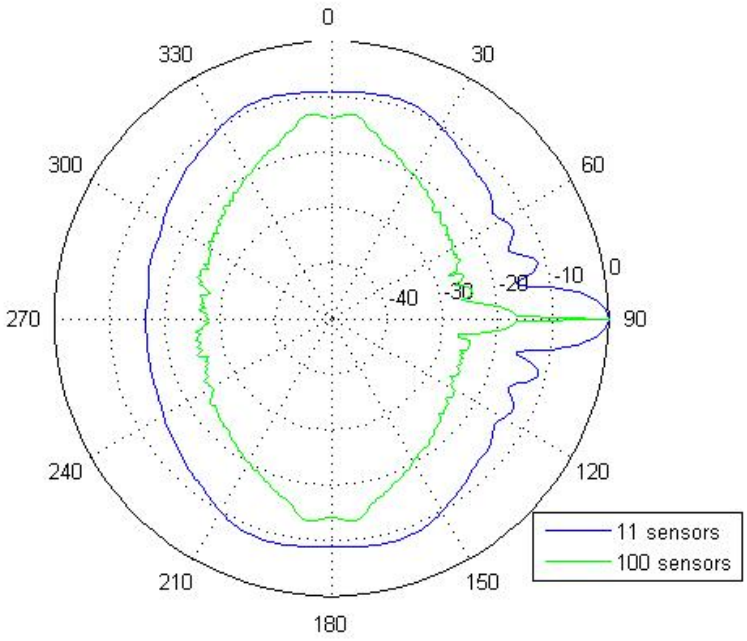

Figure 45 Beampattern Comparison with Cardioid Processing

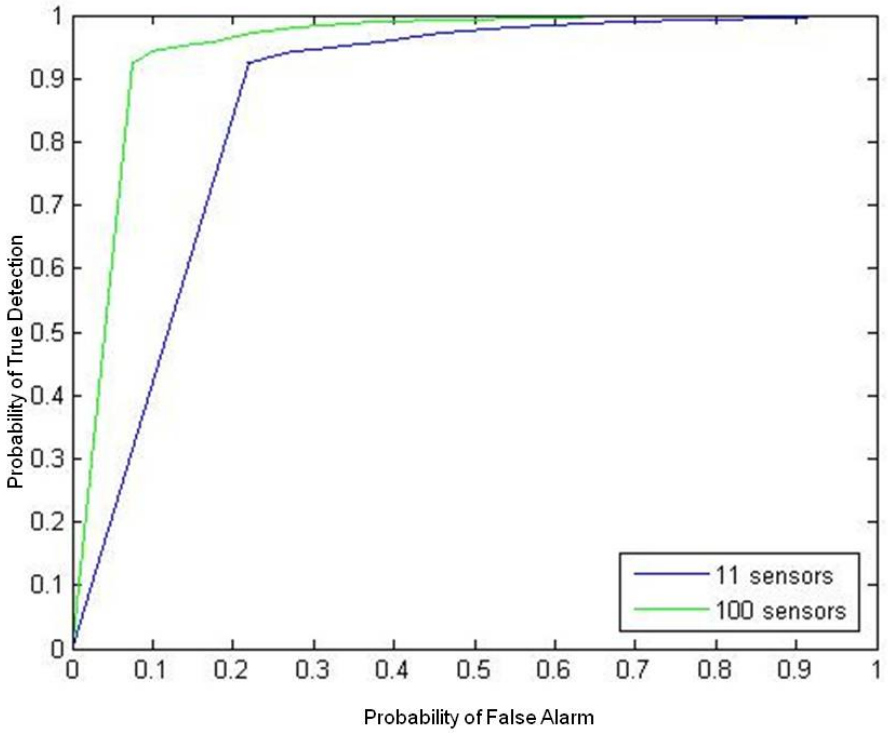

Figure 46 ROC Curve Comparison with Cardioid Processing

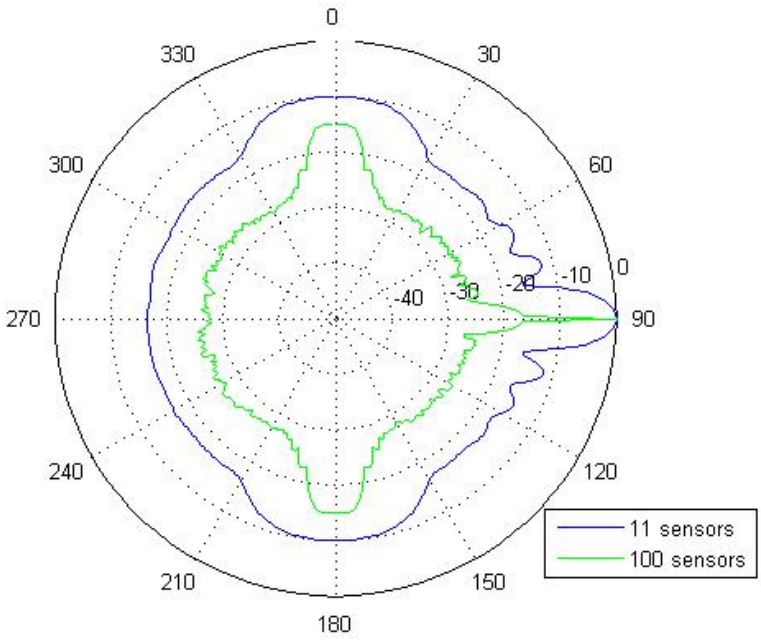

Figure 47 Beampattern Comparison with Cardynull Processing

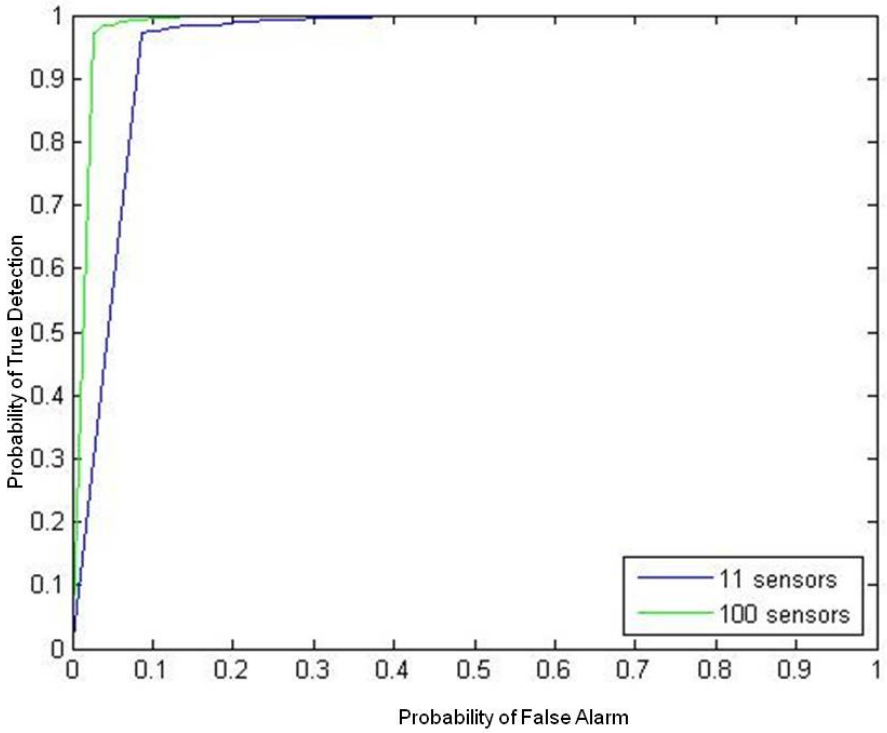

Figure 48 ROC Curve Comparison with Cardynull Processing

V. CONCLUSIONS AND RECOMMENDATIONS

A. PERFORMANCE IMPROVEMENT

Previous results (Cray & Nuttall, 2001) discussed the directivity advantages of linear arrays of vector sensors compared with conventional arrays utilizing pressure sensors. This thesis confirms the improved performance of vector sensor arrays over the pressure sensor arrays in the presence of noise. Further simulation of vector sensor arrays using linear cardioid processing and nonlinear cardynull processing indicates the potential for an additional 3 dB improvement using the nonlinear cardynull technique when the target is not near endfire and SNR is sufficient (e.g., see Figure 33).

B. LIMITATION NEAR ENDFIRE

The cardynull processing technique discussed in this paper becomes ineffective as the signal approaches endfire. Consequently, it is recommended that composite processing technique be used: for signals up to and including 45 degrees from broadside the cardynull processing should be used; for signals located greater than 45 degrees from broadside the cardioid processing should be used.

C. NAVAL APPLICATIONS

The processing techniques discussed in this thesis involve processing techniques that require minimal processing capabilities as compared to most modern data-adaptive, nonlinear processing techniques. It is anticipated that naval platforms with sufficient processing capabilities will continue to use these modern adaptive processing techniques due to their effectiveness. The combined cardynull/cardioid processing technique recommended in this thesis is, therefore, most appropriate for use on smaller platforms (e.g., UUVs) that lack the processing power of the larger platforms (e.g., surface ships).

D. FUTURE WORK

1. Other Nonlinear Processing Techniques

The nonlinear cardynull processing technique discussed in this thesis seems promising; however, it is not the only nonlinear processing technique that has been proposed by researchers. Another class of nonlinear processing techniques called hippioids has been discussed in previous articles (Smith & Leijen, 2007). Future exploration of beamformers based on these simple, non-adaptive processing techniques may prove useful. More significant improvements in processing are expected when utilizing sophisticated, nonlinear data-adaptive methods, particularly in cases of very low SNR; however, these adaptive methods also require additional computational complexity.

2. Multiple Targets

This thesis was limited to the simulation of one correlated signal (i.e., one target) in uncorrelated noise. Future work could consider the simulation of several signals in uncorrelated noise.

3. Broadband

For simplicity, this thesis considered a continuous wave (CW) with a constant frequency. Future work could consider other waveforms including broadband signals and broadband noise.

APPENDIX A. PRIMARY PROGRAM (MATLAB)

```
%% Program ROCmaker2
clc;
clear;
close all;
%% Variables:
SNRn = [-3 0 3];              % Signal to Noise Ratios (dB)
Npw = 1000;                   % Number of pressure waves
distrib  = 'normal';          % Amplitude distibution 'normal' or 'uniform'
N = 11;                       % Number of elements in the array
lambda = 150;                 % Wavelength (mm)
phiSigDeg = 90;                   % Location of signal, phi (deg)
nMC = 1000;                       % Number of Monte Carlo Runs
SAR = 1;                          % Steering Angle Resolution (deg)
tol = 15;                         % Tolerance(deg) for true target location
phiS = 0:SAR:360-SAR;             % Steering Angle, phi (deg)
thetaS = zeros(1,length(phiS));   % Steering Angle, theta (deg)
rho = 1000;                       % Density of medium (kg/m^3)
c   = 1500;                       % Speed of light in medium (m/s)
levelMultiplier = [0.05 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 0.97 0.98 ...
                                                        0.99 0.995 1 1.01];
phiSignal=phiSigDeg.*ones(1,Npw);         % Location of signal, phi (deg)
thetaSignal=zeros(1,Npw);                 % Location of signal, theta (deg)
phiNoise  = 0:(360/Npw):360-(360/Npw);    % Location of noise (deg)
thetaNoise=180*rand(1,Npw)-90;            % Location of noise, theta (deg)
mcSbeamPatterns =  zeros(3,nMC,360/SAR);
mcNbeamPatterns =  zeros(3,nMC,360/SAR);
mcSNbeamPatterns = zeros(3,nMC,360/SAR);
aveSbeamPattern =  zeros(3,360/SAR);
aveNbeamPattern =  zeros(3,360/SAR);
aveSNbeamPattern = zeros(3,360/SAR);
mcNbeamPattern = zeros(nMC,360/SAR);
mcSNbeamPattern = zeros(nMC,360/SAR);
SNRarray = zeros(1,nMC);
measuredSNR = zeros(1,3);

FPrate = zeros(length(SNRn),length(levelMultiplier),3);
TPrate = zeros(length(SNRn),length(levelMultiplier),3);
fpRate = zeros(length(levelMultiplier),3);
tpRate = zeros(length(levelMultiplier),3);
%% Create and show array arrangement
n = makeMatrix_n2(N);
r = arraySensorLocation1(n,lambda);
if 1 % plotting off
  showArrayArrangement1(r);
  showSignalLocation2(N,lambda,phiSignal);  % Illustrates signal location
  showArrayArrangement2(r);
  showNoiseLocation3(N,lambda,phiNoise,thetaNoise);
end                                          % Illustrates noise location
%% Main Program
for nn = 1:length(SNRn)
SNR = SNRn(1,nn);             % Signal to Noise Ratio
% Calculate beampatterns for each wavetype and sensor type.
[SbeamPatterns NbeamPatterns SNbeamPatterns]= ...
      beamPatternFunction5(SNR,N,Npw,distrib,nMC,lambda,phiSignal,...
```

```
                        r,thetaSignal,phiNoise,thetaNoise,phiS,thetaS,rho,c);
 for sT = 1:3
  mcSbeamPatterns(sT,1:nMC,:)  =   SbeamPatterns(sT,1:nMC,:);
  mcNbeamPatterns(sT,1:nMC,:)  =   NbeamPatterns(sT,1:nMC,:);
  mcSNbeamPatterns(sT,1:nMC,:) =  SNbeamPatterns(sT,1:nMC,:);
  aveSbeamPattern(sT,:)  =         SbeamPatterns(sT,nMC+1,:);
  aveNbeamPattern(sT,:)  =         NbeamPatterns(sT,nMC+1,:);
  aveSNbeamPattern(sT,:) =        SNbeamPatterns(sT,nMC+1,:);
 end
 clear SbeamPatterns
 clear NbeamPatterns
 clear SNbeamPatterns
% Display beampatterns and calculate true/false positive rates
for sT = 1:3
 if 1
   plotBeampatterns4(N,SNR,phiSigDeg,phiS,sT,aveSbeamPattern(sT,:),...
                                                  aveNbeamPattern(sT,:));
 end
 [fp_rate tp_rate] = rateCalc1(mcSbeamPatterns,mcNbeamPatterns,...
     mcSNbeamPatterns,aveSbeamPattern,aveNbeamPattern,aveSNbeamPattern,...
     N,phiSigDeg,phiS,SAR,tol,levelMultiplier,sT,nMC);
 if 0 % individual ROC curves turned off
   figure;plot(fp_rate,tp_rate);
   title([num2str(N) ' sensor ' sensorType ' array with signal at ' ...
                                          ', SNR = ' num2str(SNR) ' dB']);
 end
   fpRate(:,sT) = fp_rate;
   tpRate(:,sT) = tp_rate;
   FPrate(nn,:,sT) = fp_rate;
   TPrate(nn,:,sT) = tp_rate;
end
% Display final ROC curves
figure;plot(fpRate,tpRate);
legend('pressure','cardioid', 'cardynull','location','southeast');
title([num2str(N) ' sensor arrays with signal at ' num2str(phiSigDeg) ...
                                          ', SNR = ' num2str(SNR) ' dB']);
end
```

APPENDIX B. LEVEL 1 FUNCTIONS (MATLAB)

Function makeMatrix_n2:

```
function n2 = makeMatrix_n2(N)
n=(-(N-1)/2:(N-1)/2)';
n2=-n; % Reverse the order since the first sensor element is positive.
```

Function arraySensorLocation1:

```
function r = arraySensorLocation1(n,lambda)
% A function to determine the location of each sensor in an array.
% This function works for arrays with either even or odd number of
% sensors.
zeroArray=zeros(length(n),1);
d=lambda/2;
rx=n*d;
r = [rx zeroArray zeroArray]; % All sensors are located on the x-axis
```

Function showArrayArrangement1:

```
function showArrayArrangement1(r)
xLocation = zeros(length(r(:,1,1)));
figure;plot(xLocation,r(:,1,1),'ok');
```

Function showSignalLocation2:

```
function showSignalLocation2(N,lambda,phiSource)
radialDistance = N*lambda/2;
dataPointVertical = radialDistance*cosd(phiSource);
dataPointHorizontal = radialDistance*sind(phiSource);
hold on;plot(dataPointHorizontal,dataPointVertical,'pk');
plotLimit = 1.5*radialDistance;
xlim([-plotLimit plotLimit])
ylim([-plotLimit plotLimit])
title('Illustration of Signal Location');
```

Function showArrayArrangement2:

```
function showArrayArrangement2(r)
xLocation = zeros(length(r(:,1,1)));
zLocation = xLocation;
figure;plot3(xLocation,r(:,1,1),zLocation,'ok');
```

Function showNoiseLocation3:

```
function showNoiseLocation3(N,lambda,phiNoise,thetaNoise)
radialDistance = N*lambda/2;
% Note: Adjustment made since MATLAB plot coodinates are different than
% North, East, Down convention commonly used in undersea work.
dataPointX = radialDistance*cosd(thetaNoise).*sind(phiNoise);
dataPointY = radialDistance*cosd(phiNoise);
dataPointZ = -radialDistance*sind(thetaNoise).*sind(phiNoise);
hold on;plot3(dataPointX,dataPointY,dataPointZ,'xr');
plotLimit = 1.5*radialDistance;
xlim([-plotLimit plotLimit])
```

```
ylim([-plotLimit plotLimit])
zlim([-plotLimit plotLimit])
title('Illustration of Noise Locations');
```

Function beamPatternFunction5

```
function [SbeamPatterns NbeamPatterns SNbeamPatterns] = ...

beamPatternFunction5(SNR,N,Npw,distrib,nMC,lambda,phiSignal,...

r,thetaSignal,phiNoise,thetaNoise,phiS,thetaS,rho,c)
beamSum = zeros(3,length(phiS));
BEAMSUMsig = zeros(3,length(phiS));
BEAMSUMnoi = zeros(3,length(phiS));
sumSignal = 0;
sumNoise  = 0;
sumSNR    = 0;
SbeamPatterns  = zeros(3,nMC+1,length(phiS));
NbeamPatterns  = zeros(3,nMC+1,length(phiS));
if N==1
  'warning: cardTable does not provide data for just one sensor'
end
if N>150
  'warning: cardTable does not provide data for more than 150 sensors'
end
%% Monte Carlo Section
for iMC = 1:nMC      % Begin Monte Carlo
  for i_wt = 1:2         % For both wavetypes (signal and noise)
    if i_wt == 1
      wavetype = 'signal';
      [pressAmpSig pressPhsSig]= generatePressAmpPhs2(Npw,distrib);
      pressAmpSig = sqrt(10^(SNR/10))*pressAmpSig;
      pressSig = pressAmpSig.*exp(-1i*pressPhsSig);
      sumSignal = sumSignal + (sum(abs(pressSig))^2)/length(pressSig);
      [velAmpSig velPhsSig] = ...

generateVelAmpPhs2(pressAmpSig,pressPhsSig,rho,c);
    end
    if i_wt == 2
      wavetype = 'noise';
      [pressAmpNoi pressPhsNoi]= generatePressAmpPhs2(Npw,distrib);
      pressNoi = pressAmpNoi.*exp(-1i*pressPhsNoi);
      sumNoise  = sumNoise  + (sum(abs(pressNoi))^2)/length(pressNoi);
      [velAmpNoi velPhsNoi] = ...

generateVelAmpPhs2(pressAmpNoi,pressPhsNoi,rho,c);
    end
    if strcmp(wavetype,'signal')
      pressAmp = pressAmpSig;
      pressPhsSource = pressPhsSig;
      velAmp = velAmpSig;
      velPhsSource = velPhsSig;
      theta = thetaSignal;
      phi = phiSignal;
    end
    if strcmp(wavetype,'noise')
```

```
      pressAmp = pressAmpNoi;
      pressPhsSource = pressPhsNoi;
      velAmp = velAmpNoi;
      velPhsSource = velPhsNoi;
      theta = thetaNoise;
      phi = phiNoise;
    end
    k = calculateWavenumber1(lambda,theta,phi);
    k_dot_r = (k*r')';
    pressPhsArray = zeros(N,Npw);
    press = zeros(N,1);
    velPhsArray = zeros(N,Npw);
    V  = calculateV1(velAmp,theta,phi);
    vxo = zeros(N,Npw);
    vyo = zeros(N,Npw);
    vzo = zeros(N,Npw);
    vx = zeros(N,1);
    vy = zeros(N,1);
    vz = zeros(N,1);
    for n = 1:N
      pressPhsArray(n,:) = pressPhsSource + k_dot_r(n,:);
      press(n) = sum(pressAmp.*exp(-1i*pressPhsArray(n,:)));
      velPhsArray(n,:) = velPhsSource + k_dot_r(n,:);
      vxo(n,:) = V(:,1)';
      vyo(n,:) = V(:,2)';
      vzo(n,:) = V(:,3)';
      vx(n) = sum(vxo(n,:).*exp(-1i*velPhsArray(n,:)));
      vy(n) = sum(vyo(n,:).*exp(-1i*velPhsArray(n,:)));
      vz(n) = sum(vzo(n,:).*exp(-1i*velPhsArray(n,:)));
    end
    Vp = press/(rho*c);
    vp = Vp;
    ks = calculateWavenumber1(lambda,thetaS,phiS);
    v = [vx vy vz vp];
    for sT = 1:3
      if sT == 1
        sensorType = 'pressure';
      end
      if sT == 2
        sensorType = 'cardioid';
      end
      if sT == 3
        sensorType = 'cardynull';
      end
      if strcmp(sensorType,'pressure')
        beamSum(1,:) =
beamSteer5(N,length(phiS),r,lambda,v,'pressure');
      else if strcmp(sensorType,'cardioid')
        beamSum(2,:) =
beamSteer5(N,length(phiS),r,lambda,v,'cardioid');
        else if strcmp(sensorType,'cardynull')
          beamSum(3,:) =
beamSteer5(N,length(phiS),r,lambda,v,'cardynull');
          end
        end
```

```
      end
    end
    if strcmp(wavetype,'signal')
      BEAMSUMsig = BEAMSUMsig+beamSum; % Sum beampatterns for each MC
run
      beamSumSig = beamSum;
      SbeamPatterns(:,iMC,:) = beamSumSig;
    end
    if strcmp(wavetype,'noise')
      BEAMSUMnoi = BEAMSUMnoi+beamSum; % Sum beampatterns for each MC
run
      beamSumNoi = beamSum;
      NbeamPatterns(:,iMC,:) = beamSumNoi;
    end
  end
  measuredSNR = sumSignal/sumNoise;        % Begin SNR calculations
  sumSNR = sumSNR + measuredSNR;
end   % End Monte Carlo Section
aveMeasuredSNR = 10*log10(sumSNR/nMC)      % End SNR calculations
SbeamPatterns(:,nMC+1,:) = BEAMSUMsig;
NbeamPatterns(:,nMC+1,:) = BEAMSUMnoi;
SNbeamPatterns = SbeamPatterns + NbeamPatterns;
```

Function plotBeampatterns4:

```
function
plotBeampatterns4(N,SNR,phiSigDeg,phiS,sT,BEAMSUMsig,BEAMSUMnoi)
 if sT == 1
    sensorType = 'pressure';
 end
 if sT == 2
    sensorType = 'cardioid';
 end
 if sT == 3
    sensorType = 'cardynull';
 end
  beamdb2sig=plotScaling5(BEAMSUMsig); % Scaling prior to plotting
  beamdb2noi=plotScaling5(BEAMSUMnoi);
  signalBeamSum = BEAMSUMsig;          % Save these for use later
  signalBeamdb2 = beamdb2sig;          %
  noiseBeamSum = BEAMSUMnoi;           % Save these for use later
  noiseBeamdb2 = beamdb2noi;           %

  %% Combining Signal and Noise
  SNbeamSum = signalBeamSum + noiseBeamSum;
  SNbeamdb2=plotScaling5(SNbeamSum);   % Scaling prior to %plotting

  %% Comparison against expectations
  if 0 %turn off expectation plotting
    if strcmp(sensorType,'pressure')
      figure;polar6(phiS*pi/180,signalBeamdb2,'k');view([90 -90]);
      [B steerAngDeg] = signalExpectation3(N,phiSigDeg,'pressure');
      Bdb2=plotScaling5(B);
      hold on; polar6(steerAngDeg*pi/180,Bdb2,'g');
      title ('Signal compared to Signal Expectation - Pressure');
      legend('signal','signal expectation','location','southeast');
```

```
      figure;polar6(phiS*pi/180,noiseBeamdb2,'r');view([90 -90]);
      [Bsum steerAngDeg] = noiseExpectation1(N);
      Bsumdb2=plotScaling5(Bsum);
      hold on; polar6(steerAngDeg*pi/180,Bsumdb2,'g');
      title ('Noise compared to Noise Expectation - Pressure');
      legend('noise','noise expectation','location','southeast');
    end
    if strcmp(sensorType,'cardioid')||strcmp(sensorType,'cardynull')
      figure;polar6(phiS*pi/180,signalBeamdb2,'k');view([90 -90]);
      [B2 steerAngDeg2] = signalExpectation3(N,phiSigDeg,sensorType);
      Bdb2=plotScaling5(B2);
      hold on; polar6(steerAngDeg2,Bdb2,'g');
      if strcmp(sensorType,'cardioid')
        title ('Signal compared to Signal Expectation- Cardioid');
      end
      if strcmp(sensorType,'cardynull')
        title ('Signal compared to Signal Expectation- Cardynull');
      end
      legend('signal','signal expectation','location','southeast');
    end
  end
  %% Plot Signal, Noise and S&N
  if 1 % turn on
    figure; polar6(phiS*pi/180,signalBeamdb2,'k');
    hold on; polar6(phiS*pi/180,noiseBeamdb2,'r');
    hold on; polar6(phiS*pi/180,SNbeamdb2,'b');
    view([90 -90]);
    title([sensorType ' array, N = ' num2str(N) ', SNR = '
num2str(SNR)...
                        ' dB, signal at: ' num2str(phiSigDeg) '
degrees']);
    legend('signal only','noise only','S&N','location','southeast');
    % Signal and Noise Combined
    figure;polar6(phiS*pi/180,SNbeamdb2,'b');view([90 -90]);
    title([sensorType ' array, N = ' num2str(N) ', SNR = '
num2str(SNR)...
                        ' dB, signal at: ' num2str(phiSigDeg) '
degrees']);
    legend('S&N','location','southeast');
  end
end
```

Function rateCalc2:

```
function [fp_rate tp_rate] =
rateCalc2(mcSbeamPatterns,mcNbeamPatterns,...

mcSNbeamPatterns,aveSbeamPattern,aveNbeamPattern,aveSNbeamPattern,...
     N,phiSigDeg,phiS,SAR,tol,levelMultiplier,sT,nMC)
 load cardTable                    % Peak response vs. Signal
Angle(N=2:150)
table(:,1) = TABLE(:,1);           % Signal Angle (0:359)
table(:,2) = TABLE(:,N);           % Peak response(for the N specified)
 if sT == 1
    sensorType = 'pressure';
 end
```

```
 if sT == 2
    sensorType = 'cardioid';
 end
 if sT == 3
    sensorType = 'cardynull';
 end
 for iMC = 1:nMC
      beamSumSig =  mcSbeamPatterns(sT,iMC,:);
      beamSumNoi =  mcNbeamPatterns(sT,iMC,:);
      totalSum   = mcSNbeamPatterns(sT,iMC,:);
      BEAMSUMsig = aveSbeamPattern(sT,:);
      BEAMSUMnoi = aveNbeamPattern(sT,:);
      maxLevel = max(totalSum);
      hypoPosN  = zeros(1,length(levelMultiplier));
      hypoPosSN = zeros(1,length(levelMultiplier));
      for m = 1:length(levelMultiplier)
        detectThresh2 = levelMultiplier*maxLevel;  % detection level
        [hypoPosN hypoPosPlotN]  =
aboveThreshold2(beamSumNoi,detectThresh2(m));
        [leftEdgesN rightEdgesN] = edgeDetect3(hypoPosN);
        if 0 %strcmp(sensorType,'pressure') &&  iMC < 15 && m==1
                                        % conditional to limit plotting
          DT = detectThresh2(m)*ones(1,length(phiS));
          figure; set(gcf,'Renderer','Opengl');
          subplot(3,2,1);plot(phiS,beamSumNoi,'r');hold on;
          title(['  N only' ' ,MC# =' num2str(iMC)]);
          subplot(3,2,1);plot(phiS,DT,'m');
          subplot(3,2,2);plot(phiS,totalSum,'b');hold on;
          subplot(3,2,2);plot(phiS,DT,'m');
          title(['       S&N' ' ,SA = ' num2str(phiSigDeg)  ...
                                          ' ,MC# =' num2str(iMC)  ]);
        end
       [hypoPosSN hypoPosPlotSN] =
aboveThreshold2(totalSum,detectThresh2(m));
       [leftEdgesSN rightEdgesSN] = edgeDetect3(hypoPosSN);
       falseBlip = NaN(1,length(phiS));
       trueBlip =  NaN(1,length(phiS));
       if max(leftEdgesN)~=0 && max(rightEdgesN)~=0
         %'N edge is defined'
         detectPairsN =
detectRegion1(leftEdgesN,rightEdgesN,hypoPosPlotN);
         slopeSignN = slopeDetermine1(detectPairsN,hypoPosPlotN);
         locIndexN =
locationMax2(detectPairsN,slopeSignN,hypoPosPlotN);
         [trueBlip falseBlip] =
sonarBlip4(locIndexN,phiSigDeg,phiS,SAR,tol,sensorType,table);
         if 0% strcmp(sensorType,'pressure') && iMC < 15 && m==1
                                        % conditional to limit plotting
          subplot(3,2,3);plot(phiS,hypoPosPlotN,'r');title('hypoPos');
       subplot(3,2,5);polar(phiS*pi/180,falseBlip,'r*');view([90 -90]);
          title('Location of false targets');
         end
       end
      if max(leftEdgesSN)~=0 && max(rightEdgesSN)~=0
        %'SN edge is defined'
```

```
        detectPairsSN =
detectRegion1(leftEdgesSN,rightEdgesSN,hypoPosPlotSN);
        slopeSignSN = slopeDetermine1(detectPairsSN,hypoPosPlotSN);
        locIndexSN =
locationMax2(detectPairsSN,slopeSignSN,hypoPosPlotSN);
        [trueBlip falseBlip] =
sonarBlip4(locIndexSN,phiSigDeg,phiS,SAR,tol,sensorType,table);
        if 0% strcmp(sensorType,'pressure') && iMC < 15 && m==1
                                          % conditional to limit plotting
          subplot(3,2,4);plot(phiS,hypoPosPlotSN,'b');title('hypoPos');
          subplot(3,2,6);polar(phiS*pi/180,trueBlip,'g*');hold on;
          subplot(3,2,6);polar(phiS*pi/180,falseBlip,'r*');view([90 -
90]);
          title('Location of true (and false) targets');
        end
      end
      if max(falseBlip) == 1
        FP(m,iMC) = 1;
      else
        FP(m,iMC) = 0;
      end
      if max(trueBlip) == 1
        TP(m,iMC) = 1;
      else
        TP(m,iMC) = 0;
      end
     end     % End Detection Threshold
 end   % End Monte Carlo
    %aveMeasuredSNR = sumSNR(nn)/nMC    % End SNR calculations
  fp_rate = zeros(length(detectThresh2),1);   % Begin fp/tp rate calcs
  tp_rate = zeros(length(detectThresh2),1);
  for m = 1:length(detectThresh2)
    fp_rate(m) = sum(FP(m,:))/nMC;
    tp_rate(m) = sum(TP(m,:))/nMC;
  end % End fp/tp rate calcs
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. LEVEL 2 FUNCTIONS (MATLAB)

Function generatePressAmpPhs2:

```
function [amp2 phase] = generatePressAmpPhs2(Npw,distrib)
%% Variables
% Input:
%   Npw      = number of pressure waves
%   distrib  = distribution type: 'uniform' or 'normal'
% Output:
%   amp2   = amplitude of the wave (squared)
%   phase = phase of the wave (at source)
%% Calculations
% Establish the phase of the pressure waves at the source.
phase = 2*pi*rand(1,Npw);
% Check that 'distrib' is valid
if strcmp(distrib,'uniform')
       %'distrib acceptable'
else
   if strcmp(distrib,'normal')
       %'distrib acceptable'
   else
       'Warning: The variable distrib must ''uniform'' or ''normal''.'
   end
end
% Establish the amplitude of the pressure wave.
if strcmp(distrib,'normal')
      amp2 = 1 + 0.33.*randn(1,Npw);  % mean(S^2) is 1
      %'mean(S^2) = ', mean(amp2)
end
if strcmp(distrib,'uniform')
      amp2 = 2.*rand(1,Npw);      % mean(S^2) is 1
      %'mean(S^2) = ', mean(amp2)
end
```

Function generateVelAmpPhs2:

```
function [velAmp velPhs]= generateVelAmpPhs2(pressAmp,pressPhs,rho,c)
velAmp = pressAmp/(rho*c);
velPhs = pressPhs;          % Pressure and Velocity are in Phase!!
```

Function calculateWavenumber1:

```
function k = calculateWavenumber1(lambda,theta,phi)
% This function computes the wavenumber vector given lambda and
% the angles (theta,phi).  This function assumed theta and phi are
%provided in degrees.

% Note: since the wavelength, lambda, has units of mm, k has units of
1/mm.
K = 2*pi/lambda;  % k = omega/c = 2*pi*f/c = 2*pi/lambda
k = zeros(length(phi),3);
k(:,1) = -K.*cosd(phi);

k(:,2) = -K.*cosd(theta).*sind(phi);
```

```
k(:,3) = -K.*sind(theta).*sind(phi);
```

Function beamSteer5:

```
function beamSum = beamSteer5(N,nSteerAngles,r,lambda,v,sensorType)
bpress = zeros(N,nSteerAngles);
b = zeros(N,nSteerAngles);
bPrime = zeros(N,nSteerAngles);
thetaS = 0;

mm=1;
for angleDeg = 0:(360/nSteerAngles):360-(360/nSteerAngles)
    phiSdeg = angleDeg;          % Steering Angle, phi-s (deg)
    phiS = phiSdeg*pi/180;       % Steering Angle, phi-s (rad)
    ks = calculateWavenumberSteering4(lambda,thetaS,phiS);
    ks_dot_r = ks*r';

  if strcmp(sensorType,'pressure') % For Pressure-only Elements - Begin
    A = 1;
    w = defineWeightingFactorsPress1(N,A);
    wvpress = w.*v;
    wvSumpress = zeros(N,1);
    for m = 1:4
         wvSumpress = wvSumpress + wvpress(:,m);
    end
    bpress(:,mm) = wvSumpress.*exp(-1i*ks_dot_r');
                                    % For Pressure-only Elements - End
  end
  if strcmp(sensorType,'cardioid') || strcmp(sensorType,'cardynull')
                                    % For Cardioid (or Cardynull) Begin
    A = 1;
    w = defineWeightingFactors3(N,A,thetaS,phiS);
    wv = w.*v;
    wvSum = zeros(N,1);
    for m = 1:4
         wvSum = wvSum + wv(:,m);
    end
    b(:,mm) = wvSum.*exp(-1i*ks_dot_r');
                                                % For Cardioid - End
  end
  if strcmp(sensorType,'cardynull')
                                            % For Cardynull - Continue
    A = calculateA(phiS); %Part 2
    w = defineWeightingFactors3(N,A,thetaS,phiS);
    wv = w.*v;
    wvSum = zeros(N,1);
    for m = 1:4
         wvSum = wvSum + wv(:,m);
    end
    bPrime(:,mm) = wvSum.*exp(-1i*ks_dot_r');
                                                % For Cardynull - End
  end
    mm = mm+1;
end

 if strcmp(sensorType,'pressure')
```

```
  % Combine the output of each element to get the beam output
  beamsumPress = combineSensors1(N,bpress,bpress,nSteerAngles);
  beamSum = beamsumPress;

 end
 if strcmp(sensorType,'cardioid')
  % Combine the output of each element to get the beam output
  beamsumCardioid = combineSensors1(N,b,b,nSteerAngles);
  beamSum = beamsumCardioid;
 end
 if strcmp(sensorType,'cardynull')
  % Combine the output of each element to get the beam output
  beamsumCardynull = combineSensors1(N,b,bPrime,nSteerAngles);
  beamSum = beamsumCardynull;
 end
```

Function plotScaling5:

```
function Bdb2 = plotScaling5(B)
B=B/(max(B));            % B becomes a ratio between 0 and 1.
Bdb=10*log10(B);         % Bdb is a value in decibels.
Bdb2 = Bdb-max(Bdb); % Bdb2 is a value in decibels with a maximum of 0.
```

Function polar6:

```
function hpol = polar6(varargin)
%POLAR  Polar coordinate plot.
%   POLAR(THETA, RHO) makes a plot using polar coordinates of
%   the angle THETA, in radians, versus the radius RHO.
%   POLAR(THETA,RHO,R) uses the radial limits specified by the two
element
%   vector R.
%   POLAR(THETA,RHO,S) uses the linestyle specified in string S.
%   See PLOT for a description of legal linestyles.
%   POLAR(THETA,RHO,R,S) uses the linestyle specified in string S and
the
%   radial limits in R.
%
%   POLAR(AX,...) plots into AX instead of GCA.
%
%   H = POLAR(...) returns a handle to the plotted object in H.
%
%   Example:
%      t = 0:.01:2*pi;
%      polar(t,sin(2*t).*cos(2*t),'--r')
%
%   See also PLOT, LOGLOG, SEMILOGX, SEMILOGY.
%
%   Revised version by Daniel Armyr, 2009. Based on Mathworks original.
%   Further revised by David Tassia, August 2009; see notes with DT
initials.
%   Additional revision by Tassia, September 2009 to eliminate the
%     label at 0 degrees (which otherwise interfers with the title).
%   Additional revision by Tassia to eliminate the restriction of the
%     length of the variable 'rho'.
```

```
%   Copyright 1984-2007 The MathWorks, Inc.
%   $Revision: 5.22.4.9 $  $Date: 2007/08/27 17:06:52 $

% Parse possible Axes input
[cax,args,nargs] = axescheck(varargin{:});
error(nargchk(1,4,nargs,'struct'));

if nargs < 1 || nargs > 4
    error('MATLAB:polar:InvalidInput', 'Requires 2 to 4 data
arguments.')
elseif nargs == 2
    theta = args{1};
    rho = args{2};
    if ischar(rho)
        line_style = rho;
        rho = theta;
        [mr,nr] = size(rho);
        if mr == 1
            theta = 1:nr;
        else
            th = (1:mr)';
            theta = th(:,ones(1,nr));
        end
    else
        line_style = 'auto';
    end
    radial_limits = [];
elseif nargs == 1
    theta = args{1};
    line_style = 'auto';
    rho = theta;
    [mr,nr] = size(rho);
    if mr == 1
        theta = 1:nr;
    else
        th = (1:mr)';
        theta = th(:,ones(1,nr));
    end
    radial_limits = [];
elseif nargs == 3
    if ( ischar(args{3}) )
        [theta,rho,line_style] = deal(args{1:3});
        radial_limits = [];
    else
        [theta,rho,radial_limits] = deal(args{1:3});
        line_style = 'auto';
        if ( ~(numel(radial_limits) == 2) )
            error ( 'R must be a 2 element vector' );
        end
    end
else %nargs == 4
    [theta,rho,radial_limits,line_style] = deal(args{1:4});
    if ( ~(numel(radial_limits) == 2) )
        error ( 'R must be a 2 element vector' );
```

```
    end
end

if ischar(theta) || ischar(rho)
    error('MATLAB:polar:InvalidInputType', 'Input arguments must be
numeric.');
end
if ~isequal(size(theta),size(rho))
    error('MATLAB:polar:InvalidInput', 'THETA and RHO must be the same
size.');
end
% DT: This section was added to facilitate plotting using the
%     otherwise standard polar.m commands

rho = rho+50;  % DT: Add 50 dB so the plotted radius can be positive
% DT: Now zero all of the data that is more than 50 dB below the
maximum.
for k = 1:length(rho)
    if rho(k) <= 0
         rho(k) = 0;
    end
end
% DT: end of added section
% get hold state
cax = newplot(cax);

next = lower(get(cax,'NextPlot'));
hold_state = ishold(cax);

% get x-axis text color so grid is in same color
tc = get(cax,'xcolor');
ls = get(cax,'gridlinestyle');

% Hold on to current Text defaults, reset them to the
% Axes' font attributes so tick marks use them.
fAngle  = get(cax, 'DefaultTextFontAngle');
fName   = get(cax, 'DefaultTextFontName');
fSize   = get(cax, 'DefaultTextFontSize');
fWeight = get(cax, 'DefaultTextFontWeight');
fUnits  = get(cax, 'DefaultTextUnits');
set(cax, 'DefaultTextFontAngle',  get(cax, 'FontAngle'), ...
    'DefaultTextFontName',   get(cax, 'FontName'), ...
    'DefaultTextFontSize',   get(cax, 'FontSize'), ...
    'DefaultTextFontWeight', get(cax, 'FontWeight'), ...
    'DefaultTextUnits','data')

% only do grids if hold is off
if ~hold_state

% make a radial grid
    hold(cax,'on');
    set(cax,'dataaspectratio',[1 1 1],'plotboxaspectratiomode','auto')

    % ensure that Inf values don't enter into the limit calculation.
```

```
    arho = abs(rho(:));
    if ( isempty(radial_limits) )
        maxrho = max(arho(arho ~= Inf));
        minrho = 0;
        hhh=line([minrho minrho maxrho maxrho],[minrho maxrho maxrho
minrho],'parent',cax);
        v = [get(cax,'xlim') get(cax,'ylim')];
        ticks = numel(get(cax,'ytick'));
        delete(hhh);
        % check radial limits and ticks
        rmin = v(1); rmax = v(4); rticks = max(ticks-1,2);
        if rticks > 5   % see if we can reduce the number
            if rem(rticks,2) == 0
                rticks = rticks/2;
            elseif rem(rticks,3) == 0
                rticks = rticks/3;
            end
        end
        rinc = (rmax-rmin)/rticks;

    else
        rmax = radial_limits(2);
        rmin = radial_limits(1);
        order = (10^floor(log10(rmax-rmin)));
        firstDigit = floor((rmax-rmin)/order);
        if ( firstDigit <= 1 )
            step = 0.2*order;
        elseif ( firstDigit <= 3 )
            step = 0.5*order;
        elseif ( firstDigit <= 7 )
            step = order;
        else
            step = 2*order;
        end
        rinc = step;
    end

% define a circle
    th = 0:pi/50:2*pi;
    xunit = cos(th);
    yunit = sin(th);
% now really force points on x/y axes to lie on them exactly
    inds = 1:(length(th)-1)/4:length(th);
    xunit(inds(2:2:4)) = zeros(2,1);
    yunit(inds(1:2:5)) = zeros(3,1);
% plot background if necessary
    if ~ischar(get(cax,'color')),
       patch('xdata',xunit*(rmax-rmin),'ydata',yunit*(rmax-rmin), ...
             'edgecolor',tc,'facecolor',get(cax,'color'),...
             'handlevisibility','off','parent',cax);
    end

% draw radial circles
    c82 = cos(82*pi/180);
    s82 = sin(82*pi/180);
```

```
    for i=(rmin+rinc):rinc:rmax
        hhh = line(xunit*(i-rmin),yunit*(i-
rmin),'linestyle',ls,'color',tc,'linewidth',1,...
                  'handlevisibility','off','parent',cax);
        text((i-2-rmin+rinc/20)*c82,(i-2-rmin+rinc/20)*s82, ...
            ['  ' num2str(i-50)],'verticalalignment','bottom',...
            'handlevisibility','off','parent',cax)
        % DT: Replacing i with i-50 provides the desired labeling
        %     (i.e. maximum of 0 dB minimum of -50 dB)
        % DT: Replacing i with i-2 moves labels (e.g. -10 dB) closer to
the
        %     circles that they are labeling.
    end
    set(hhh,'linestyle','-') % Make outer circle solid

% plot spokes
    th = (1:6)*2*pi/12;
    cst = cos(th); snt = sin(th);
    cs = [-cst; cst];
    sn = [-snt; snt];
    line((rmax-rmin)*cs,(rmax-
rmin)*sn,'linestyle',ls,'color',tc,'linewidth',1,...
         'handlevisibility','off','parent',cax)

% annotate spokes in degrees
    rt = 1.1*(rmax-rmin);
    for i = 1:length(th)
        text(rt*cst(i),rt*snt(i),int2str(i*30),...
             'horizontalalignment','center',...
             'handlevisibility','off','parent',cax);
        if i == length(th)
            loc = int2str(0);
        else
            loc = int2str(180+i*30);
        end
        if i ~= length(th)  %DT: This statement eliminates the label at
0 degrees
             text(-rt*cst(i),-
rt*snt(i),loc,'horizontalalignment','center',...
             'handlevisibility','off','parent',cax)
        end
    end

% set view to 2-D
    view(cax,2);
% set axis limits
    axis(cax,(rmax-rmin)*[-1 1 -1.15 1.15]);

    setappdata( cax, 'rMin', rmin );

else
    %Try to find the inner radius of the current axis.
    if ( isappdata ( cax, 'rMin' ) )
        rmin = getappdata( cax, 'rMin' );
```

```
    else
        rmin = 0;
    end
end

% Reset defaults.
set(cax, 'DefaultTextFontAngle', fAngle , ...
    'DefaultTextFontName',   fName , ...
    'DefaultTextFontSize',   fSize, ...
    'DefaultTextFontWeight', fWeight, ...
    'DefaultTextUnits',fUnits );

% transform data to Cartesian coordinates.
xx = (rho - rmin).*cos(theta);
yy = (rho - rmin).*sin(theta);

% plot data on top of grid
if strcmp(line_style,'auto')
    q = plot(xx,yy,'parent',cax);
else
    q = plot(xx,yy,line_style,'parent',cax);
end

if nargout == 1
    hpol = q;
end

if ~hold_state
    set(cax,'dataaspectratio',[1 1 1]), axis(cax,'off');
set(cax,'NextPlot',next);
end
set(get(cax,'xlabel'),'visible','on')
set(get(cax,'ylabel'),'visible','on')

if ~isempty(q) && ~isdeployed

makemcode('RegisterHandle',cax,'IgnoreHandle',q,'FunctionName','polar')
;
end
```

Function signalExpectation3:

```
function [B steerAngles] = signalExpectation3(N,angleSource,sensorType)
%% Function signalExpectation2
%  This function provides input and output for functions that provide
the
%  expected beampattern for one signal.  The functions called include:
%     a) equation2131e: simulates pressure sensors using Van Trees
equation 2.131.
%     b) cardioidBeamPatt1: simulates cardioid sensors
%     c) cardynullBeamPatt1: simulates cardynull sensors
%  The output is the beampattern for an array steered around 360
degrees.
%  This program provides the output expected for a Signal located at
```

```
%  the angle specified in thetaSource.  The results can be used for
%  comparison with more complicated beamforming routines.
%
%% Variables:
%      steerAngles =  Steering angles (deg)
%   Inputs:
%      N           =  Number of (pressure only) sensors in the linear
array
%      angleSource =  Source angle (deg)
%   Outputs:
%      B           =  Beampattern (dB)
%      steerAngles  =  An array of steering angles (deg)
%%
if strcmp(sensorType,'pressure')
  steerAngles = 0:0.25:359.75;
  B = equation2131e(N,angleSource,steerAngles);
end
if strcmp(sensorType,'cardioid')
  [B steerAngles] = cardioidBeamPatt2(N,angleSource,0);
                                          % 0 assumes signal in z = 0
plane
end
if strcmp(sensorType,'cardynull')
    [B steerAngles] = cardynullBeamPatt2(N,angleSource,0);
                                          % 0 assumes signal in z = 0
plane
End
```

Function noiseExpection1:

```
function [Bsum thetaSteer] = noiseExpectation1(N)
%% Function noiseExpectation1
%  This program provides a composite plot by calling
%  the function equation2131e for each source angle (0:0.25:359.75) and
%  summing the result.  The resulting beam pattern is what we would
%expect if noise was coming from all directions.
%  The results can be used for comparison with more complicated
%beamforming routines.
%% Variables:
%   Inputs:
%      N           =  Number of (pressure only) sensors in the linear
array
%   Outputs:
%      B           =  Beampattern (dB)
%      thetaSteer  =  An array of steering angles (deg)
%%
thetaSteer = 0:0.25:359.75;
Bsum = zeros(1,length(thetaSteer));
for thetaSource = 0:0.25:359.75;
   Bsum = Bsum+equation2131e(N,thetaSource,thetaSteer);
End
```

Function aboveThreshold2:

```
function [hypoPos hypoPosPlot] = aboveThreshold2(RL,detectThresh)
```

```
hypoPos = zeros(1,length(RL));
hypoPosPlot = hypoPos;
for m = 1:length(RL)
    if RL(m) >= detectThresh
        hypoPos(m) = 1;
        hypoPosPlot(m) = RL(m);
    else
       %hypoPos(m) already equals 0;
       %hypoPosPlot(m) already equals 0;
    end
end
```

Function edgeDetect3:

```
function [leftEdges rightEdges] = edgeDetect3(hypoPos)
nl = 1;
nr = 1;
% for the majority of the array
for m = 1:length(hypoPos)-1
  if hypoPos(m)==0 && hypoPos(m+1)==1
    leftEdges(nl) = m+1;
    nl = nl+1;
  end
  if hypoPos(m)==1 && hypoPos(m+1)==0
    rightEdges(nr) = m;
    nr = nr+1;
  end
end
% for the ends of the array
if hypoPos(length(hypoPos))==0 && hypoPos(1)==1
    leftEdges(nl) = 1;
end
if hypoPos(length(hypoPos))==1 && hypoPos(1)==0
    rightEdges(nr) = length(hypoPos);
end
if nl == 1   % if the left edge still hasn't been defined
    leftEdges(nl) = 0;  % assign zero
end
if nr == 1   % if the right edge still hasn't been defined
    rightEdges(nr) = 0; % assign zero
end
```

Function detectRegion1:

```
function detectPairs = detectRegion1(leftEdges,rightEdges,hypoPosPlot)
% This function simply creates a matrix that contains the indices of
%the left side and the right side of each detection region.
detectPairs = zeros(length(leftEdges),2);
 for n = 1:length(leftEdges)   % for each left edge
  m = leftEdges(n);           % use the index of the left edge
  small = length(hypoPosPlot);
  for k = 1:length(rightEdges)% and find the index of the next right
edge
     if rightEdges(k) >= leftEdges(n)
       if rightEdges(k)-leftEdges(n) <= small
         mm = rightEdges(k);
         small = rightEdges(k)-leftEdges(n);
       end
     end
  end
  if leftEdges(n) > max(rightEdges)  %Special case when beam strattles
zero degrees
     mm = min(rightEdges);
  end
 detectPairs(n,1) = m;
 detectPairs(n,2) = mm;
 end
end
```

Function slopeDetermine1:

```
function slopeSign = slopeDetermine1(detectPairs,hypoPosPlot)
M = length(detectPairs(:,2));
index = 1;
slopeSign = NaN(1,360);
for m = 1:M
  kl = detectPairs(m,1);
  kr = kl+1;
  if  detectPairs(m,1) <= detectPairs(m,2) % Majority case
    difference = detectPairs(m,2)-detectPairs(m,1);
    for nn = 1:difference
      while kl <= detectPairs(m,2)                % Define slope signs
        if kr > length(hypoPosPlot)
          kr = 1;
        end
        if kl > length(hypoPosPlot)
          kl = 1;
        end
        slopeSign(kl) = sign(hypoPosPlot(kr)-hypoPosPlot(kl));
        kl = kl+1;
        kr = kr+1;
      end
    end
  end
  if detectPairs(m,1) > detectPairs(m,2)        %Special case when beam
```

```
    mmm = detectPairs(m,2) + length(hypoPosPlot);  %strattles zero
%degrees
    difference = mmm - detectPairs(m,1);
    for nn = 1:difference
      while index < mmm
        if kr > length(hypoPosPlot)
           kr = 1;
        end
        if kl > length(hypoPosPlot)
          kl = 1;
        end
        slopeSign(kl) = sign(hypoPosPlot(kr)-hypoPosPlot(kl)); %Define
        index = index+1;                                       %slope
%signs
        kl = kl+1;
        kr = kr+1;
      end
    end
  end
end
%figure; plot(1:360,slopeSign,'b-'); axis([0 360 -1.2 1.2]);
End
```

Function locationMax2:

```
function maxIndices = locationMax2(detectPairs,slopeSign,hypoPosPlot)
%This function finds the approximate location of each maximum and
outputs
%the index for this location.  It does this for all maximums(0:359
degrees).
if length(slopeSign) ~= length(hypoPosPlot)
  'warning: the variables slopeSign and hypoPosPlot do not have the
same length'
end
M = length(detectPairs(:,2));
ki = 1;
for m = 1:M                 % for each detection region
  if detectPairs(m,1) < detectPairs(m,2)
    difference = detectPairs(m,2)-detectPairs(m,1);
  else
    if detectPairs(m,1) > detectPairs(m,2)
      difference = 360+detectPairs(m,2)-detectPairs(m,1);
    else
      difference = 0;
      maxIndices(ki) = detectPairs(m,1);
      ki =ki+1;
    end
  end
  kl = detectPairs(m,1);
  kr = kl+1;
  for mm = 1:difference
    if kr > length(slopeSign)
      kr = 1;
    end
    if kl > length(slopeSign)
```

```
      kl = 1;
    end
    if slopeSign(kl)==1 && slopeSign(kr)==-1       % a maximum is near
%here
      ks = kr;
      maxIndices(ki) = ks;
      ki =ki+1;
    end
    if detectPairs(m,2)-detectPairs(m,1) == 1      % a maximum is near
%here
      ks = kl;
      maxIndices(ki) = ks;
      ki =ki+1;
    end
    if slopeSign(detectPairs(m,1)) == -1           % a maximum is near
%here
      ks = kl;
      maxIndices(ki) = ks;
      ki =ki+1;
    end
    kl = kl+1;
    kr = kr+1;
  end
end
```

Function sonarBlip4:

```
function [trueBlip falseBlip] =
sonarBlip4(locationIndex,phiSignal,phiSteer,SAR,tol,sensorType,table)
% First determine the index for the angle where phiSignal and phiSteer
% are the same.
M = length(phiSteer);
if strcmp(sensorType,'cardioid')
  for m = 1:M
    if phiSteer(m) == phiSignal
        mm = m;
    end
  end
  % Now 'mm' is the index at which phiSteer(mm) is equal to phiSignal.
  % In other words, 'mm' is the index that gives the response where the
  % signal is located.
end
if strcmp(sensorType,'pressure')
  if SAR ~= 1
    'warning: sonarBlip4 assumes SAR = 1 for pressure-only arrays'
  end
  for m = 1:M
    if phiSteer(m) == phiSignal
        mm = m;
        if mm==1
          mmm = 1;
        else
          mmm = 362 - mm;
        end
    end
```

```
  end
  % Now 'mm' is the index at which phiSteer(mm) is equal to phiSignal.
  % In other words, 'mm' is the index that gives the response where the
  % signal is located.
end
if strcmp(sensorType,'cardynull')
  if SAR ~= 1
    'warning: sonarBlip4 assumes SAR = 1 for cardynull processor'
  end
  for m = 1:360
     if table(m,1) == phiSignal
        mm = table(m,2)+1;
     end
  end
end
    % For cardynull processing 'mm' is index where the maximum response
%is expected.  Since the cardynull is nonlinear the peak response is
    % closer to broadside than the actual signal location.

% The location of each contact is given by locationIndex.
% Create a new arrays with the location of each true/false contact.
trueBlip = NaN(1,M);
falseBlip = NaN(1,M);
% tol = angle tolerance (degrees) {for determining if true target}
% SAR = steering angle resolution (degrees)
for n = 1:length(locationIndex)
  if(locationIndex(n)>=mm-
round(tol/SAR))&&(locationIndex(n)<=mm+round(tol/SAR))
      trueBlip(locationIndex(n)) = 1;
  else
    if strcmp(sensorType,'pressure') && ...
      (locationIndex(n)>=mmm-
round(tol/SAR))&&(locationIndex(n)<=mmm+round(tol/SAR))
         trueBlip(locationIndex(n)) = 1;
    else
         falseBlip(locationIndex(n)) = 1;
    end
  end
end
```

APPENDIX D. LEVEL 3 FUNCTIONS

Function calculateWavenumberSteering4:

```
function k = calculateWavenumberSteering4(lambda,theta,phi)
% This function computes the wavenumber vector given lambda and
% the angles (theta,phi).
% k = omega/c = 2*pi*f/c = 2*pi/lambda
% Note: since the wavelength, lambda, has units of mm, k has units of
%1/mm.
K = 2*pi/lambda;
k = zeros(1,3);
k(1,1) = K*cos(phi);
k(1,2) = K*cos(theta)*sin(phi);
k(1,3) = K*sin(theta)*sin(phi);
% The following section to eliminate small residual values for
% cos(pi/2), etc.
eps = 1E-6;
for m = 1:3
    if abs(k(1,m)) <= eps
         k(1,m) = 0;
    end
end
```

Function defineWeightingFactorsPress1:

```
function w = defineWeightingFactorsPress1(N,A)
%wh=hann(N);             % Weighting factors (Hanning)
wu=1/N*ones(N,1);        % Weighting factors (Uniform)
wx=zeros(N,1);
wy=zeros(N,1);
wz=zeros(N,1);
wp=wu*A;
w=[wx wy wz wp];
% The following section eliminates small residual values for
% cos(pi/2), etc.
eps = 1E-15;
for m = 1:4
    if abs(w(:,m)) <= eps
         w(:,m) = 0;
    end
end
```

Function defineWeightingFactors3:

```
function w = defineWeightingFactors3(N,A,thetaS,phiS)
%wh=hann(N);             % Weighting factors (Hanning)
wu=1/N*ones(N,1);        % Weighting factors (Uniform)
wx=wu*cos(phiS);
wy=wu*cos(thetaS)*sin(phiS);
wz=wu*sin(thetaS)*sin(phiS);
wp=wu*A;
w=[wx wy wz wp];
% The following section eliminates small residual values for
% cos(pi/2), etc.
```

```
eps = 1E-15;
for m = 1:4
    if abs(w(:,m)) <= eps
         w(:,m) = 0;
    end
end
```

Function calculateA:

```
function A = calculateA(phiS)
A = -cos(2*phiS);
```

Function combineSensors1:

```
function beamSum  = combineSensors1(N,b1,b2,nSteeringAngles)
% This function combines the output of each vector sensor to get
% the total beam.
bA = zeros(1,nSteeringAngles);
bB = zeros(1,nSteeringAngles);
for n=1:N  % Accumulate the contribution of each sensor
    bA(1,:) = bA(1,:) + b1(n,:);
    bB(1,:) = bB(1,:) + b2(n,:);
end
beamSum = abs(bA).*abs(bB);
if N == 1
    beamSum = beamSum.^2;
end
```

Function equation2131e:

```
function B = equation2131e(N,thetaSource,thetaSteer)
% This function completes the calculation of equation 2.131 of
%vanTrees.
% The output is in power or intensity units (i.e. the square of the
% result of the equation).
%
% Inputs:
%      N =        Number of (pressure-only) sensors in the linear array
%      thetaSource =  Source angle (deg)
%      thetaSteer  =  Steering angle (deg)
%
dol = 0.5;                % d over lamdba
    B= 1/N.*...
      (sin(pi*N*dol*(cos(thetaSource*pi/180)-
cos(thetaSteer*pi/180)))).^2./...
      (sin(pi*dol*(cos(thetaSource*pi/180)-cos(thetaSteer*pi/180))));
% Since the previous statement evaluates to 0/0 when the source and
% steering angle are equal, the following conditional statement was
%added.
for n = 1:length(thetaSteer)
    if (cos(thetaSource*pi/180)-cos(thetaSteer(n)*pi/180)) == 0
        B(n) = 1;
    end
end
B=B.^2;
```

Function cardioidBeamPatt2:

```
function [B phi] = cardioidBeamPatt2(N,phiSource,thetaSource)
% This function is based on the equations provided in the program
% "beampatterns" by K.B. Smith.
% It provides the beampattern for an array of vector sensors.

m=(1:N);  % N is # of array elements

S1=1; S2=0; S3=0;  % Relative signal receive level of 3 signals
phi1=phiSource*pi/180; phi2=0*pi/180; phi3=0*pi/180;   % Angle relative
to end-fire of 3 signals

theta1=thetaSource*pi/180; theta2=0*pi/180; theta3=0*pi/180;  %
Relative phase (radians) of 3 signals

theta=0; phi=(0:359)*pi/180;  % Steering angles

press=zeros(N,length(phi)); card=press; dynnull=press;

for el=1:N,
    phs1=i*pi*(el-1)*cos(phi1); phs2=i*pi*(el-1)*cos(phi2);
phs3=i*pi*(el-1)*cos(phi3);  % Plane wave phases of 3 signals at array
elements locations
    A=1; B=1; C=1; D=1;  % Equal weighting scheme (Cardioid)
    press(el,:)=S1*exp(phs1);  % Processing on pressure signal only

card(el,:)=A*(S1*cos(phi1)*exp(phs1)+S2*cos(phi2)*exp(phs2)+S3*cos(phi3
)*exp(phs3))*cos(phi)+...

B*(S1*cos(theta1)*sin(phi1)*exp(phs1)+S2*cos(theta2)*sin(phi2)*exp(phs2
)+S3*cos(theta3)*sin(phi3)*exp(phs3))*cos(theta)*sin(phi)+...

C*(S1*sin(theta1)*sin(phi1)*exp(phs1)+S2*sin(theta2)*sin(phi2)*exp(phs2
)+S3*sin(theta3)*sin(phi3)*exp(phs3))*sin(theta)*sin(phi)+...
       D*(S1*exp(phs1)+S2*exp(phs2)+S3*exp(phs3))*cos(0*phi);  %
Processing on vector signals using cardioid weighting
    Ddyn=-cos(2*phi);  % Weighting scheme for Dynamic Null steering

dynnull(el,:)=A*(S1*cos(phi1)*exp(phs1)+S2*cos(phi2)*exp(phs2)+S3*cos(p
hi3)*exp(phs3))*cos(phi)+...

B*(S1*cos(theta1)*sin(phi1)*exp(phs1)+S2*cos(theta2)*sin(phi2)*exp(phs2
)+S3*cos(theta3)*sin(phi3)*exp(phs3))*cos(theta)*sin(phi)+...

C*(S1*sin(theta1)*sin(phi1)*exp(phs1)+S2*sin(theta2)*sin(phi2)*exp(phs2
)+S3*sin(theta3)*sin(phi3)*exp(phs3))*sin(theta)*sin(phi)+...
       Ddyn.*(S1*exp(phs1)+S2*exp(phs2)+S3*exp(phs3)).*cos(0*phi);  %
Processing on vector signals using dynamic null weighting

end

bpress=zeros(length(phi),1); bcard=bpress; bdynnull=bpress;
for n=1:length(phi),
```

```
    rep=exp(i*pi*cos(phi(n)).*(m-1));  % Plane wave phase replica
across array elements

    tmp=xcorr(press(:,n),rep);
    bpress(n)=abs(tmp(max(2,N)));

    tmp=xcorr(card(:,n),rep);
    bcard(n)=abs(tmp(max(2,N)));

    tmp=xcorr(dynnull(:,n),rep);
    bdynnull(n)=abs(tmp(max(2,N)));
end
B = bcard;
B = B.^2;
B = B';
```

Function cardynullBeamPatt2:

```
function [B phi] = cardynullBeamPatt2(N,phiSource,thetaSource)
% This function is based on the equations provided in the program
% "beampatterns" by K.B. Smith.
% It provides the beampattern for cardynull processing of an array of
% vector sensors.

m=(1:N);  % N is # of array elements

S1=1; S2=0; S3=0;  % Relative signal receive level of 3 signals
phi1=phiSource*pi/180; phi2=0*pi/180; phi3=0*pi/180;   % Angle relative
to end-fire of 3 signals

theta1=thetaSource*pi/180; theta2=0*pi/180; theta3=0*pi/180;  %
%Relative phase (radians) of 3 signals

theta=0; phi=(0:359)*pi/180;  % Steering angles

press=zeros(N,length(phi)); card=press; dynnull=press;

for el=1:N,
    phs1=i*pi*(el-1)*cos(phi1); phs2=i*pi*(el-1)*cos(phi2);
phs3=i*pi*(el-1)*cos(phi3);  % Plane wave phases of 3 signals at array
%elements locations
    A=1; B=1; C=1; D=1;  % Equal weighting scheme (Cardioid)
    press(el,:)=S1*exp(phs1);  % Processing on pressure signal only

card(el,:)=A*(S1*cos(phi1)*exp(phs1)+S2*cos(phi2)*exp(phs2)+S3*cos(phi3
)*exp(phs3))*cos(phi)+...

B*(S1*cos(theta1)*sin(phi1)*exp(phs1)+S2*cos(theta2)*sin(phi2)*exp(phs2
)+S3*cos(theta3)*sin(phi3)*exp(phs3))*cos(theta)*sin(phi)+...

C*(S1*sin(theta1)*sin(phi1)*exp(phs1)+S2*sin(theta2)*sin(phi2)*exp(phs2
)+S3*sin(theta3)*sin(phi3)*exp(phs3))*sin(theta)*sin(phi)+...
```

```
        D*(S1*exp(phs1)+S2*exp(phs2)+S3*exp(phs3))*cos(0*phi);  %
%Processing on vector signals using cardioid weighting
    Ddyn=-cos(2*phi);  % Weighting scheme for Dynamic Null steering

dynnull(el,:)=A*(S1*cos(phi1)*exp(phs1)+S2*cos(phi2)*exp(phs2)+S3*cos(p
hi3)*exp(phs3))*cos(phi)+...

B*(S1*cos(theta1)*sin(phi1)*exp(phs1)+S2*cos(theta2)*sin(phi2)*exp(phs2
)+S3*cos(theta3)*sin(phi3)*exp(phs3))*cos(theta)*sin(phi)+...

C*(S1*sin(theta1)*sin(phi1)*exp(phs1)+S2*sin(theta2)*sin(phi2)*exp(phs2
)+S3*sin(theta3)*sin(phi3)*exp(phs3))*sin(theta)*sin(phi)+...
        Ddyn.*(S1*exp(phs1)+S2*exp(phs2)+S3*exp(phs3)).*cos(0*phi);  %
%Processing on vector signals using dynamic null weighting

end

bpress=zeros(length(phi),1); bcard=bpress; bdynnull=bpress;
for n=1:length(phi),
    rep=exp(i*pi*cos(phi(n)).*(m-1));  % Plane wave phase replica
%across array elements

    tmp=xcorr(press(:,n),rep);
    bpress(n)=abs(tmp(max(2,N)));

    tmp=xcorr(card(:,n),rep);
    bcard(n)=abs(tmp(max(2,N)));

    tmp=xcorr(dynnull(:,n),rep);
    bdynnull(n)=abs(tmp(max(2,N)));
end
B = bcard.*bdynnull;
B = B';
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. PROGRAM AND FUNCTION TO DETERMINE LOCATION OF CARDYNULL MAXIMUM RESPONSE (MATLAB)

```
%% Program cardynullTablemaker
% For a Cardynull processor, this program calculates the angle of the
% maximum (peak) response for all signal angles.
% The results are provided in a table (called 'TABLE') that lists the
% signal angle and the corresponding peak response (to the nearest 1
%deg.)
% for line arrays with 2-150 sensors.  The results are stored in
% 'cardTable' which can be loaded whenever the peak response for the
% nonlinear Cardynull processor is needed.
% Caution: Peak responses are not accurate near endfire (i.e. for 351-9
% degrees and for 171-189 degrees).

TABLE = zeros(360,150);
for N = 2:150
  table = cardynullTablemakerFunc2(N);
  if N == 2
    TABLE(:,1) = table(:,1);
  end
  TABLE(:,N) = table(:,2);
end
save cardTable TABLE

%% cardynullTablemakerFunc2
% For a Cardynull processor, this function calculates the angle of the
% maximum (peak) response for all signal angles.
% The results are provided in a table that lists the signal angle and
%the corresponding peak response (to the nearest 1 degree).
function table = cardynullTablemakerFunc1(N)
  sensorType = 'cardynull';
  angSig = (0:359)';
  angMaxResponse = angSig;
  for m = 11:171
    phiSigDeg = m-1;
    [B2 ~] = signalExpectation3(N,phiSigDeg,sensorType);
    Bdb2=plotScaling5(B2);
    maxBdb2 = max(Bdb2);
    for n = 0:359
      if Bdb2(n+1) == maxBdb2
        angMaxResponse(m) = n;
      end
    end
  end
  n=11;
  for m = 351:-1:191
    angMaxResponse(m) = 360-angMaxResponse(n);
    n=n+1;
  end
  table = [angSig angMaxResponse];
end
% Note: for signal angles near endfire (i.e. for angles (351 - 9) and
```

```
% (171 -189).  The maximum response angle (angMaxResponse) is set to
%equal to the signal angle (angSig); in other words: table(m,1) =
%table(m,2)
% near endfire.  It is anticipated that studies will be completed
% between 10-170 degrees and 190-350 degrees for the Cardynull
%processor
% since directing the null in the ambiguous direction would cause
%problems
% when the ambiguous direction approaches the signal direction (i.e.
near
% endfire).
```

LIST OF REFERENCES

Beranek, L. L. (1986). Acoustics, *American Institute of Physics*, 161–163.

Birdall, T. G. (1973, January). *Theory of signal detectability–ROC curves and their character*. ONR-TR-177, Office of Naval Research.

Cox, H., & Zeskind, R. M. (1992, October 26–28). *Twenty-Sixth Asilomar conference on signals, systems and computers*. Monterey, CA.

Cray, B. A., & Nuttall, A. H. (2001). Directivity factors for linear arrays of velocity sensors, *J. Acoust. Soc. Am*. 110, 324–331.

Cray, B. A. (1992, April 9). *Acoustic vector sensor,* United States Patent, Patent No. US 6370084.

Cray, B. A., & Evora, V. F. (2004, February 24). *Highly directive underwater acoustic receiver*, United States Patent, Patent No. US 6697302.

Green, D. M., & Swets, J. A. (1966). *Signal detection theory and psychophysics*, John Wiley and Sons, Inc. (Reprinted by Robert E. Krieger Publishing Co. Inc., 1974).

Hayes, H. C. (1920, March 19). Detection of submarines, *Proc. Amer. Phil., Soc*., *I.XIX, A*.

Kinsler, L. E., Frey, A. R., Coppens, A. B., & Sanders. J. V. (1976). *Fundamentals of acoustics* (4th ed.). John Wiley and Sons, Inc.

Meier, L. (2000, September 29). *An analysis of vector acoustics single sensor cross correlation detection performance*. (Unpublished report).

Meier, L. (2001, May 17). *Optimization of detection with vector sensors*. (Unpublished report).

Meier III, L., & Cray, B. A. (2001, November 5). *Monte Carlo simulation of vector sensor performance: A Broadband signal in isotropic noise*. NUWC-NPT Technical Document 11323, Naval Undersea Warfare Center, Newport Division.

Peterson, W. W., & Birdall, T. G. (1953). The theory of signal detectibility. *Univ. of Mich. Eng. Res. Inst*. Rep. 13.

Smith, K. B. (2007, May 1). *Vector sensor geometries and array symmetries*. (Unpublished notes).

Smith, K. B., & Leijen, A. V. (2007, July). Steering vector sensor array elements with linear cardioids and nonlinear hippioids. *J. Acoustic Soc. Am*. *122*(1).

Urick, R. J. (1983, April). *Principles of underwater sound* (3rd ed.). McGraw Hill.

van Trees, H. L. (2002). *Detection, estimation, and modulation theory, Part iv, optimum array processing*. John Wiley & Sons.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Technical Library
   Naval Undersea Warfare Center Division Newport
   Newport, Rhode Island

4. Prof. Kevin B. Smith
   Naval Postgraduate School
   Monterey, California

5. Code 01CTO (Anthony Ruffa)
   Naval Undersea Warfare Center Division Newport
   Newport, Rhode Island

6. Code 1511 (Lewis Meier)
   Naval Undersea Warfare Center Division Newport
   Newport, Rhode Island

7. Code 1513 (Joseph Dibiase)
   Naval Undersea Warfare Center Division Newport
   Newport, Rhode Island

8. Code 821 (Benjamin A. Cray)
   Naval Undersea Warfare Center Division Newport
   Newport, Rhode Island

9. Code 321MS (Mike Traweek)
   Office of Naval Research
   Arlington, Virginia